Compressonator Documentation

Release Compressonator - latest build

AMD Developer Tools

Jul 10, 2023

Content

Compressonator SDK source URL: https://github.com/GPUOpen-Tools/Compressonator

Compressonator is a set of tools to allow artists and developers to more easily work with compressed assets and easily visualize the quality impact of various compression technologies. It consists of a GUI application, a command line application, and an SDK for easy integration into a developer toolchain.

A variety of block based codecs, palletized and grayscale encoders, and useful tools for generating mip-maps, comparing the quality of compressed and uncompressed images and batch-compressing large databases of images are included.

Developers and texture artists can optimize the level of quality and performance that best suits the requirements for their game assets and pipeline. Textures are compressed according to specific data streaming requirements, balancing both power and memory of the targeted devices while maintaining quality that users expect.

The most popular codecs are provided in a comprehensive easy to use library with a simple interface that can be integrated into game engines and tools.

Compressonator supports Microsoft Windows®, Linux and Mac builds.

Developer SDK

1.1 CMP Core

This library supports the following codecs BC1 to BC7, also known as ATI1N, ATI2N and DXTC.

The main API call for both compression and decompression is at the block level for each of these codecs.

1.1.1 GPU Shaders

With OpenCL (OCL) and DirectX (DXC): These API are availble for use in user defined shaders by including BCn_Encode_Kernel.h and Common_Def.h

```
CGU_Vec2ui CompressBlockBC1_UNORM( CMP_IN CGU_Vec3f BlockRGB[16], CMP_IN CGU_FLOAT
↪fquality,CGU_BOOL isSRGB)
CGU_Vec4ui CompressBlockBC2_UNORM( CMP_IN CGU_Vec3f BlockRGB[16], CMP_IN CGU_FLOAT
↪BlockA[16], CGU_FLOAT fquality, CGU_BOOL isSRGB)
CGU_Vec4ui CompressBlockBC3_UNORM( CMP_IN CGU_Vec3f BlockRGB[16], CMP_IN CGU_FLOAT
↪BlockA[16], CGU_FLOAT fquality,CGU_BOOL isSRGB)
CGU_Vec2ui CompressBlockBC4_UNORM( CMP_IN CGU_FLOAT Block[16], CGU_FLOAT fquality)
CGU_Vec4ui CompressBlockBC5_UNORM( CMP_IN CGU_FLOAT BlockU[16], CMP_IN CGU_FLOAT
↪BlockV[16], CGU_FLOAT fquality)
```

For OCL addition API is provided under a generic API interface for all BCn encoders: CMP_STATIC CMP_KERNEL void CMP_GPUEncoder(. . .) see plugin CMP_GPU_OCL for details on how this is used. For DXC CMP_Core contains sample HLSL shaders used by the plugin CMP_GPU_DXC

1.1.2 Error Codes

All Core API calls return a int success 0 (CMP_CORE_OK) or error value > 0 (CMP_CORE_ERR) for a more detailed and up to date list look at the file Common_Def.h

```
CGU_CORE_OK = 0,                              // No errors, call was successfull
CGU_CORE_ERR_UNKOWN,                          // An unknown error occurred
CGU_CORE_ERR_NEWMEM,                          // New Memory Allocation Failed
CGU_CORE_ERR_INVALIDPTR,                      // The pointer value used is invalid or null
CGU_CORE_ERR_RANGERED,                        // values for Red   Channel is out of range␣
→(too high or too low)
CGU_CORE_ERR_RANGEGREEN,                      // values for Green Channel is out of range␣
→(too high or too low)
CGU_CORE_ERR_RANGEBLUE,                       // values for Blue  Channel is out of range␣
→(too high or too low)
```

1.1.3 Codec Quality Settings

BC1, BC2, and BC3 have discrete quality settings, These settings are available in the following ranges (varying the q setting in these ranges will have no new effects, q is a discrete coarse setting)

```
q = 0.0   to 0.01 sets lowest quality and fast compression
q = 0.101 to 0.6 sets mid-quality
q = 0.601 to 1.0 set the best quality and low performance
BC4 and BC5 have no quality settings, no changes in quality will occur if set.
BC6 & BC7 & ASTC have full q ranges from 0 to 1.0
```

1.1.4 Create and Destroy Options Pointers

Context based reference pointers are used instead of class or structure definitions for each codec, it provides a clear definition of what the setting options are for each codec and a means for easy expansion of the features for future releases.

BCn is used to describe a short form for all of the codecs BC1 to BC7.

All BCn codecs will use default max quality settings when a null pointer is used for the CompressBlock calls, users can create multiple contexts to set preferred Quality, Mode Masks , Channel Mapping, Etc. . .

A void pointer reference is used to relay the setting options to the respected setting and encoding functions. When a context pointer is created for a particular codecs like BC1 its should not be used for another codecs like BC2, it will create unpredictable results, runtime exceptions or errors.

```
int CMP_CDECL CreateOptionsBC1(void **optionsBC1);
int CMP_CDECL CreateOptionsBC2(void **optionsBC2);
int CMP_CDECL CreateOptionsBC3(void **optionsBC3);
int CMP_CDECL CreateOptionsBC4(void **optionsBC4);
int CMP_CDECL CreateOptionsBC5(void **optionsBC5);
int CMP_CDECL CreateOptionsBC6(void **optionsBC6);
int CMP_CDECL CreateOptionsBC7(void **optionsBC7);
```

These calls removes the context memory used by the CreateOptions.

```
int CMP_CDECL DestroyOptionsBC1(void *optionsBC1);
int CMP_CDECL DestroyOptionsBC2(void *optionsBC2);
int CMP_CDECL DestroyOptionsBC3(void *optionsBC3);
int CMP_CDECL DestroyOptionsBC4(void *optionsBC4);
int CMP_CDECL DestroyOptionsBC5(void *optionsBC5);
int CMP_CDECL DestroyOptionsBC6(void *optionsBC6);
int CMP_CDECL DestroyOptionsBC7(void *optionsBC7);
```

1.1.5 Channel Weights

Setting channel Weights : Applies to BC1, BC2 and BC3 valid ranges are [0..1.0f] Default is {1.0f, 1.0f, 1.0f}

With swizzle formats the weighting applies to the data within the specified channel not the channel itself

```
int CMP_CDECL SetChannelWeightsBC1(void *options, float WeightRed, float WeightGreen,
→float WeightBlue);
int CMP_CDECL SetChannelWeightsBC2(void *options, float WeightRed, float WeightGreen,
→float WeightBlue);
int CMP_CDECL SetChannelWeightsBC3(void *options, float WeightRed, float WeightGreen,
→float WeightBlue);
```

1.1.6 Quality Settings

All values are clamped in the range 0.0 to 1.0, the encoding performance is much slower when quality is set high.

```
int CMP_CDECL SetQualityBC1(void *options, float fquality);
int CMP_CDECL SetQualityBC2(void *options, float fquality);
int CMP_CDECL SetQualityBC3(void *options, float fquality);
int CMP_CDECL SetQualityBC4(void *options, float fquality);
int CMP_CDECL SetQualityBC5(void *options, float fquality);
int CMP_CDECL SetQualityBC6(void *options, float fquality);
int CMP_CDECL SetQualityBC7(void *options, float fquality);
```

1.1.7 Alpha Threshold

BC1 uses 1 bit alpha for encoding. This bit is set when pixel values are >= alphaThreshold, default is 128

```
int CMP_CDECL SetAlphaThresholdBC1(void *options, unsigned char alphaThreshold);
```

1.1.8 Mode Masks

This mask can be used to enable any or all encoding modes of the codec. If all are enabled the resulting quality of the image is increased, reducing the modes enabled lowers the quality and increases performance.

mask : BC6 uses 14 compression modes, default is all enabled (0x3FFF)

mask : BC7 uses 8 compression modes, default is all enabled (0xFF)

The current BC6H codec always defaults all modes enabled. Next release will enable this feature setting!

BC7 Mask values are mapped by bit position with mode 0 been at the lowest significant bit (LSB) value in the mask.

MSB LSB								
mask bits	1	1	1	1	1	1	1	1
BC7 mode	7	6	5	4	3	2	1	0

Example: To enable modes 6 & 1 only the mask value is set to hex 0x42 or binary 01000010 if mask is set to <= 0 then default will be 0xCF which is what prior versions of Compressonator used.

```
int CMP_CDECL SetMaskBC6(void *options, unsigned int  mask);
int CMP_CDECL SetMaskBC7(void *options, unsigned char mask);
```

1.1.9 Decoder Channel Mapping

The channel mapping can be set for BC1, BC2 and BC3 decoders to decode channels Red(0) ,Green(1), Blue(2) and Alpha(3) as RGBA channels [0,1,2,3] (default True) else BGRA maps to [0,1,2,3] In this release bool is used as a swizzle setting (Red, Blue channel swap). Future release will allow different channel mapping!

```
int CMP_CDECL SetDecodeChannelMapping(void *options, bool mapRGBA);
```

1.1.10 BC7 Alpha Options

imageNeedsAlpha : Reserved for future use (default is false)

colourRestrict : Sets block to avoid using Combined Alpha modes. value of true enables else false disables the setting (default is false)

alphaRestrict : Avoid blocking issues with punch-through or threshold alpha encoding. value of true enables else false disables the setting (default is false)

Each pixel in the tile is checked for alpha values, if any alpha value in the input block is set below 255 (blockNeedsAlpha = TRUE) or is a value of {0 or 255} (blockAlphaZeroOne = TRUE), respective modes will be either be kept or discarded using the ModeMask as a guide and the following conditions for each block the Mode cycles from 0 to 7.

If the block needs alpha and this mode doesn't support alpha then indicate that this is not a valid mode for the block: if (blockNeedsAlpha == TRUE) and (Mode has NO ALPHA) then that Mode is disabled from been used for processing

Optional user restriction for color only blocks so that they do not use modes that have combined color and alpha : this avoids the possibility that the encoder might choose an alpha other than 255 (due to parity) and cause something to become accidentally slightly transparent it's possible that when encoding 3-component texture applications will assume that the 4th component can safely be assumed to be 255 all the time

if (blockNeedsAlpha == TRUE) and (Mode has COMBINED ALPHA) and (ColourRestrict == TRUE) then that Mode is excluded for processing

Optional restriction for blocks with alpha to avoid issues with punch-through or threshold alpha encoding:

if (blockNeedsAlpha == TRUE) and (Mode has COMBINED ALPHA) and (AlphaRestrict == TRUE) and (blockAlphaZeroOne == TRUE) then that Mode is excluded for processing

```
int CMP_CDECL SetAlphaOptionsBC7(void *options, bool imageNeedsAlpha, bool␣
→colourRestrict, bool alphaRestrict);
```

1.1.11 BC7 Error Threshold

The **minThreshold** (default 5.0f) and **maxThreshold** (default 80.0f) is used to map the quality setting and partition ranges used during quantization. These values should be set prior to using the SetQualityBC7 call else it will have no effect and the defaults will be used.

To increase performance, adjust maxThreshold higher, to reduce performance and increase quality adjust maxThreshold lower.

minThreshold is used to clamp a minimum error value that is added to maxThreshold when quality is set to max 1.0;

The upper range values are not checked, values are clampped to 0.0 if negative!

```
int CMP_CDECL SetErrorThresholdBC7(void *options, float minThreshold, float␣
→maxThreshold);
```

1.1.12 Compressing Blocks

srcBlock : Buffer pointer reference to a source block to use for compression, the source buffer is expected to be in RGBA:8888 format for BC1..5,BC7 codecs. For BC6H the buffer should be in Half float format.

srcStrideInBytes : Is the number of bytes required to access the next row of the 4x4 block reference from the original srcBlock pointer.

srcStrideInShorts : Is the number of short int values required to access the next row of the 4x4 block reference from the original srcBlock pointer.

cmpBlock : Pointer reference to a destination block that is typically in the range of 8 to 16 bytes for BCn codecs.

options : Is the encoder reference created by CreateOptions. This can be a null pointer, which uses the codecs default settings for high quality encoding.

BC4 codec uses a srcBlock array of 16 bytes. This array typically represents the Red Channel of a RGBA_8888 4x4 image block.

BC5 codec uses two srcBlocks each are a array of 16 bytes. These two arrays typically represents the Red and Green channels of a RGBA_8888 4x4 image block.

```
int CMP_CDECL CompressBlockBC1(unsigned char *srcBlock, unsigned int ⌴
→srcStrideInBytes, unsigned char cmpBlock[8 ], void *options CMP_DEFAULTNULL);
int CMP_CDECL CompressBlockBC2(unsigned char *srcBlock, unsigned int ⌴
→srcStrideInBytes, unsigned char cmpBlock[16], void *options CMP_DEFAULTNULL);
int CMP_CDECL CompressBlockBC3(unsigned char *srcBlock, unsigned int ⌴
→srcStrideInBytes, unsigned char cmpBlock[16], void *options CMP_DEFAULTNULL);
int CMP_CDECL CompressBlockBC4(unsigned char *srcBlock, unsigned int ⌴
→srcStrideInBytes, unsigned char cmpBlock[8], void *options  CMP_DEFAULTNULL);
int CMP_CDECL CompressBlockBC5(unsigned char *srcBlock1, unsigned int⌴
→srcStrideInBytes1,
                              unsigned char *srcBlock2, unsigned int⌴
→srcStrideInBytes2,
                              unsigned char cmpBlock[16], void *options  CMP_
→DEFAULTNULL);
int CMP_CDECL CompressBlockBC6(unsigned short *srcBlock, unsigned int⌴
→srcStrideInShorts, unsigned char cmpBlock[16], void *options CMP_DEFAULTNULL);
int CMP_CDECL CompressBlockBC7(unsigned char *srcBlock, unsigned int ⌴
→srcStrideInBytes, unsigned char cmpBlock[16], void *options CMP_DEFAULTNULL);
```

1.1.13 Decompressing Blocks

```
int CMP_CDECL DecompressBlockBC1(unsigned char cmpBlock[8 ], unsigned char⌴
→srcBlock[64] , void *options CMP_DEFAULTNULL);
int CMP_CDECL DecompressBlockBC2(unsigned char cmpBlock[16], unsigned char⌴
→srcBlock[64] , void *options CMP_DEFAULTNULL);
int CMP_CDECL DecompressBlockBC3(unsigned char cmpBlock[16], unsigned char⌴
→srcBlock[64] , void *options CMP_DEFAULTNULL);
int CMP_CDECL DecompressBlockBC4(unsigned char cmpBlock[8 ], unsigned char⌴
→srcBlock[16] , void *options  CMP_DEFAULTNULL);
int CMP_CDECL DecompressBlockBC5(unsigned char cmpBlock[16], unsigned char⌴
→srcBlock1[16], unsigned char srcBlock2[16], void *options  CMP_DEFAULTNULL);
int CMP_CDECL DecompressBlockBC6(unsigned char cmpBlock[16], unsigned short⌴
→srcBlock[48], void *options CMP_DEFAULTNULL);
int CMP_CDECL DecompressBlockBC7(unsigned char cmpBlock[16], unsigned char⌴
→srcBlock[64] , void *options CMP_DEFAULTNULL);
```

1.1.14 Example Usage of Core API

Sample application to process a 4x4 image block using Compress and Decompress API's SDK files required for application to build:

CMP_Core.h CMP_Core_xx.lib For static libs xx is either MD, MT or MDd or MTd, when using DLL's make sure the CMP_Core_xx_DLL.dll is in exe path

Example usage is shown as below, error checking on the function returns has been omitted for clarity:

```
// BC1 options context
void *BC1Options;

// Create an options context reference using this call, it returns a pointer to use
↪for BC1 codec settings
// All functions used in CMP Core return error codes to check if calls were successful
CreateOptionsBC1(&BC1Options);

// Check if pointer is allocated, can also use the call function return CGU_CORE_OK
if (BC1Options == NULL) {
    printf("Failed to create BC1 Options Context!");
    return (-1);
}

// Setting Channel Weights {Red,Green,Blue}
SetChannelWeightsBC1(BC1Options, 0.3086f, 0.6094f, 0.0820f);

// Compress a sample image shape0 which is a 4x4 RGBA_8888 block, its stride is 16
↪bytes to the next row of the 4x4 image block.
// Users can use a pointer to any sized image buffers to reference a 4x4 block by
↪supplying a stride offset for the next row.

unsigned char shape0_RGBA[64] = { filled with image source data as RGBA ...};

// cmpBuffer is a byte array of 8 byte to hold the compressed results.
unsigned char cmpBuffer[8]   = { 0 };

// Compress the source into cmpBuffer
CompressBlockBC1(shape0_RGBA, 16, cmpBuffer, BC1Options);

// Example to decompress comBuffer back to a RGBA_8888 4x4 image block
unsigned char imgBuffer[64] = { 0 };
DecompressBlockBC1(cmpBuffer,imgBuffer,BC1Options);

// Can compare the original image (shape0_RGBA) vs the decompressed image (imgBuffer)
↪for quality

// Remove the Options Setting Memory
DestroyOptionsBC1(BC1Options);
```

A full example project is provided here

- core_example demonstrates compression and decompression of all of the available codecs with various quality and performance settings.

The example is also distributed through CompressonatorCore installer in the release page.

1.2 CMP Framework

This library depends only on standard libaray.

1.2.1 CMP Error Codes

```
typedef enum {
CMP_OK = 0,                               // Ok.
CMP_ABORTED,                              // The conversion was aborted.
CMP_ERR_INVALID_SOURCE_TEXTURE,          // The source texture is invalid.
CMP_ERR_INVALID_DEST_TEXTURE,            // The destination texture is invalid.
CMP_ERR_UNSUPPORTED_SOURCE_FORMAT,       // The source format is not a supported format.
CMP_ERR_UNSUPPORTED_DEST_FORMAT,         // The destination format is not a supported
↪format.
CMP_ERR_UNSUPPORTED_GPU_ASTC_DECODE,     // The gpu hardware is not supported.
CMP_ERR_UNSUPPORTED_GPU_BASIS_DECODE,    // The gpu hardware is not supported.
CMP_ERR_SIZE_MISMATCH,                   // The source and destination texture sizes do
↪not match.
CMP_ERR_UNABLE_TO_INIT_CODEC,            // Compressonator was unable to initialize the
↪codec needed for conversion.
CMP_ERR_UNABLE_TO_INIT_DECOMPRESSLIB,    // GPU_Decode Lib was unable to initialize the
↪codec needed for decompression .
CMP_ERR_UNABLE_TO_INIT_COMPUTELIB,       // Compute Lib was unable to initialize the
↪codec needed for compression.
CMP_ERR_CMP_DESTINATION,                 // Error in compressing destination texture
CMP_ERR_MEM_ALLOC_FOR_MIPSET,            // Memory Error: allocating MIPSet compression
↪level data buffer
CMP_ERR_UNKNOWN_DESTINATION_FORMAT,      // The destination Codec Type is unknown! In
↪SDK refer to GetCodecType()
CMP_ERR_FAILED_HOST_SETUP,               // Failed to setup Host for processing
CMP_ERR_PLUGIN_FILE_NOT_FOUND,           // The required plugin library was not found
CMP_ERR_UNABLE_TO_LOAD_FILE,             // The requested file was not loaded
CMP_ERR_UNABLE_TO_CREATE_ENCODER,        // Request to create an encoder failed
CMP_ERR_UNABLE_TO_LOAD_ENCODER,          // Unable to load an encode library
CMP_ERR_NOSHADER_CODE_DEFINED,           // No shader code is available for the
↪requested framework
CMP_ERR_GPU_DOESNOT_SUPPORT_COMPUTE,     // The GPU device selected does not support
↪compute
CMP_ERR_NOPERFSTATS,                     // No Performance Stats are available
CMP_ERR_GPU_DOESNOT_SUPPORT_CMP_EXT,     // The GPU does not support the requested
↪compression extension!
CMP_ERR_GAMMA_OUTOFRANGE,                // Gamma value set for processing is out of
↪range
CMP_ERR_PLUGIN_SHAREDIO_NOT_SET,         // The plugin C_PluginSetSharedIO call was not
↪set and is required for this plugin to operate
CMP_ERR_UNABLE_TO_INIT_D3DX,             // Unable to initialize DirectX SDK or get a
↪specific DX API
CMP_ERR_GENERIC                          // An unknown error occurred.
} CMP_ERROR;
```

1.2.2 Kernel Options and Extensions

```
typedef enum CMPComputeExtensions {
    CMP_COMPUTE_FP16 = 0x0001,          ///< Enable Packed Math Option for GPU
    CMP_COMPUTE_MAX_ENUM = 0x7FFF
} CMP_ComputeExtensions;

/// An enum selecting the different GPU driver types.
typedef enum {
    CMP_CPU = 0,    //Use CPU Only, encoders defined CMP_CPUEncode or Compressonator␣
↪lib will be used
    CMP_HPC = 1,    //Use CPU High Performance Compute Encoders with SPMD support␣
↪defined in CMP_CPUEncode)
    CMP_GPU = 2,    //Use GPU Kernel Encoders to compress textures using Default GPU␣
↪Framework auto set by the codecs been used
    CMP_GPU_OCL = 3,    //Use GPU Kernel Encoders to compress textures using OpenCL␣
↪Framework
    CMP_GPU_DXC = 4,    //Use GPU Kernel Encoders to compress textures using DirectX␣
↪Compute Framework
    CMP_GPU_VLK = 5     //Use GPU Kernel Encoders to compress textures using Vulkan␣
↪Compute Framework
} CMP_Compute_type;


struct KernalOptions {
    CMP_ComputeExtensions  Extensions; // Compute extentions to use, set to 0␣
↪(default) if you are not using any extensions
    CMP_DWORD  height;                  // Height of the encoded texture.
    CMP_DWORD  width;                   // Width of the encoded texture.
    CMP_FLOAT  fquality;                // Set the quality used for encoders 0.05 is␣
↪the lowest and 1.0 for highest.
    CMP_FORMAT format;                  // Encoder codec format to use for processing
    CMP_Compute_type encodeWith;        // Host Type : default is HPC, options are␣
↪[HPC or GPU (reserved for future use)]
    CMP_INT    threads;                 // requested number of threads to use␣
↪(1=single) max is 128 for HPC, 0 for Auto
};

typedef enum _CMP_ANALYSIS_MODES
{
    CMP_ANALYSIS_MSEPSNR = 0x00000000  // Enable Measurement of MSE and PSNR for 2␣
↪mipset image samples
} CMP_ANALYSIS_MODES;

typedef struct
{
    unsigned long analysisMode;     // Bit mapped setting to enable various forms of␣
↪image anlaysis
    unsigned int  channelBitMap;    // Bit setting for active channels to do analysis␣
↪on and reserved features
                                    // msb(....ABGR)lsb
    double        mse;      // Mean Square Error for all active channels in a given␣
↪CMP_FORMAT
    double        mseR;     // Mean Square for Red Channel
    double        mseG;     // Mean Square for Green
    double        mseB;     // Mean Square for Blue
    double        mseA;     // Mean Square for Alpha
    double        psnr;     // Peak Signal Ratio for all active channels in a given␣
↪CMP_FORMAT
```

```
    double        psnrR;    // Peak Signal Ratio for Red Chennel
    double        psnrG;    // Peak Signal Ratio for Green
    double        psnrB;    // Peak Signal Ratio for Blue
    double        psnrA;    // Peak Signal Ratio for Alpha
} CMP_AnalysisData;
```

1.2.3 Encoder Settings

```
// The structure describing block encoder level settings.
typedef struct {
    unsigned int  width;   // Width of the encoded texture.
    unsigned int  height;  // Height of the encoded texture.
    unsigned int  pitch;   // Distance to start of next line..
    float         quality; // Set the quality used for encoders 0.05 is the lowest␣
→and 1.0 for highest.
    unsigned int  format;  // Format of the encoder to use: this is a enum set see␣
→compressonator.h CMP_FORMAT
} CMP_EncoderSetting;
```

1.2.4 Mip Map Interfaces

```
// MIP MAP Interfaces
CMP_INT CMP_MaxFacesOrSlices(const CMP_MipSet* pMipSet, CMP_INT nMipLevel);
CMP_INT CMP_API CMP_CalcMinMipSize(CMP_INT nHeight, CMP_INT nWidth, CMP_INT␣
→MipsLevel);

CMP_VOID CMP_API CMP_FreeMipSet(CMP_MipSet *MipSetIn);
CMP_VOID CMP_API CMP_GetMipLevel(CMP_MipLevel *data, const CMP_MipSet* pMipSet, CMP_
→INT nMipLevel, CMP_INT nFaceOrSlice);
CMP_INT  CMP_API CMP_CalcMaxMipLevel(CMP_INT nHeight, CMP_INT nWidth, CMP_BOOL␣
→bForGPU);
CMP_INT  CMP_API CMP_CalcMinMipSize(CMP_INT nHeight, CMP_INT nWidth, CMP_INT␣
→MipsLevel);

CMP_INT  CMP_API CMP_GenerateMIPLevels(CMP_MipSet *pMipSet, CMP_INT nMinSize);
CMP_INT  CMP_API CMP_GenerateMIPLevelsEx(CMP_MipSet* pMipSet, CMP_CFilterParams*␣
→pCFilterParams);

CMP_ERROR CMP_API CMP_CreateCompressMipSet(CMP_MipSet* pMipSetCMP, CMP_MipSet*␣
→pMipSetSRC);

// MIP Map Quality
CMP_UINT  CMP_API CMP_getFormat_nChannels(CMP_FORMAT format);
CMP_ERROR CMP_API CMP_MipSetAnlaysis(CMP_MipSet* src1, CMP_MipSet* src2, CMP_INT␣
→nMipLevel, CMP_INT nFaceOrSlice, CMP_AnalysisData* pAnalysisData);
```

1.2.5 User Processing Callback

```
// CMP_MIPFeedback_Proc
// Feedback function for conversion.
```

```
// \param[in] fProgress The percentage progress of the texture compression.
// \param[in] mipProgress The current MIP level been processed, value of fProgress =␣
↪mipProgress
// \return non-NULL(true) value to abort conversion
typedef bool(CMP_API* CMP_MIPFeedback_Proc)(CMP_MIPPROGRESSPARAM mipProgress);
```

1.2.6 Texture Load and Save

Has complete support for dds file format and limited versions of jpeg, png, bmp, hdr, psd, tga, gif, pic, psd, pgm and ppm file formats as used in std_image.h

```
//---------------------------------------------
// CMP_Compute Lib: Texture Encoder Interfaces
//---------------------------------------------
CMP_ERROR   CMP_API CMP_LoadTexture(const char *sourceFile, CMP_MipSet *pMipSet);
CMP_ERROR   CMP_API CMP_SaveTexture(const char *destFile,   CMP_MipSet *pMipSet);
```

1.2.7 Texture Processing

Two types of texture processing API are provided.

CMP_ProcessTexture is a higher level call that sets up a complete framework and supports textures generated with mip level processing. this api uses the alternate framework API's and CMP_ProcessTexture.

CMP_CompressTexture is a lower level access API that users can use to setup processing textures. By default processing uses the Compressonator Core codecs with a CPU framework.

```
CMP_ERROR   CMP_API CMP_ProcessTexture(CMP_MipSet* srcMipSet, CMP_MipSet* dstMipSet,␣
↪KernalOptions kernelOptions,  CMP_Feedback_Proc pFeedbackProc);
CMP_ERROR   CMP_API CMP_CompressTexture(KernalOptions *options,CMP_MipSet srcMipSet,␣
↪CMP_MipSet dstMipSet,CMP_Feedback_Proc pFeedback);
```

1.2.8 Using Alternate Frameworks

These options provides user options to set setup the CMP_CompressTexture interface pipeline for CPU, HPC or GPU based processing, with the "encodeWith" option in KernelOptions.

```
//---------------------------------------------
// CMP_Compute Lib: Host level interface
//---------------------------------------------
CMP_ERROR CMP_API CMP_CreateComputeLibrary(CMP_MipSet *srcTexture, KernelOptions ␣
↪*kernelOptions, void *Reserved);
CMP_ERROR CMP_API CMP_DestroyComputeLibrary(CMP_BOOL forceClose);
CMP_ERROR CMP_API CMP_SetComputeOptions(ComputeOptions *options);
```

1.2.9 Block level Access

Provides users with options to process any image block by providing a pointer to the source textures and destination buffers to be processed. The source pointer can be any location on the original texture as long as it is bounded within a valid 4x4 image block. The destination buffer must also be sufficiently large enough to hold the compressed buffer generated by the target format.

```
//----------------------------------------------------------
// Generic API to access the core using CMP_EncoderSetting
//----------------------------------------------------------
CMP_ERROR CMP_API CMP_CreateBlockEncoder(void **blockEncoder, CMP_EncoderSetting
→encodeSettings);
void       CMP_API CMP_DestroyBlockEncoder(void  **blockEncoder);

CMP_ERROR CMP_API CMP_CompressBlock(void  **blockEncoder,void *srcBlock, unsigned int
→sourceStride, void *dstBlock, unsigned int dstStride);
CMP_ERROR CMP_API CMP_CompressBlockXY(void  **blockEncoder,unsigned int blockx,
→unsigned int blocky, void *imgSrc, unsigned int sourceStride, void *cmpDst,
→unsigned int dstStride);
```

1.2.10 Format and Processor Utils

```
CMP_VOID   CMP_API CMP_Format2FourCC(CMP_FORMAT format,   CMP_MipSet *pMipSet);
CMP_FORMAT CMP_API CMP_ParseFormat(char* pFormat);
CMP_INT    CMP_API CMP_NumberOfProcessors();
```

1.2.11 Framework Example: Mip Level Processing

You will need to include a header file and a lib file: **CMP_Framework** and **CMP_Framework_MD.lib**

```
const char*      pszSourceFile = argv[1];
const char*      pszDestFile   = argv[2];
CMP_FORMAT       destFormat    = CMP_ParseFormat(argv[3]);
CMP_ERROR        cmp_status;
CMP_FLOAT        fQuality;

try {
  fQuality = std::stof(argv[4]);
  if (fQuality < 0.0f) {
    fQuality = 0.0f;
    std::printf("Warning: Quality setting is out of range using 0.0\n");
  }
  if (fQuality > 1.0f) {
    fQuality = 1.0f;
    std::printf("Warning: Quality setting is out of range using 1.0\n");
  }
} catch (...) {
  std::printf("Error: Unable to process quality setting\n");
  return -1;
}

if (destFormat == CMP_FORMAT_Unknown) {
    std::printf("Error: Unsupported destination format\n");
    return 0;
}

//---------------
// Load the image
//---------------
CMP_MipSet MipSetIn;
memset(&MipSetIn, 0, sizeof(CMP_MipSet));
```

(continues on next page)

```cpp
cmp_status = CMP_LoadTexture(pszSourceFile, &MipSetIn);
if (cmp_status != CMP_OK) {
    std::printf("Error %d: Loading source file!\n",cmp_status);
    return -1;
}

//----------------------------------------------------------------------
// generate mipmap level for the source image, if not already generated
//----------------------------------------------------------------------

if (MipSetIn.m_nMipLevels <= 1)
{
    CMP_INT requestLevel = 10; // Request 10 miplevels for the source image

    //------------------------------------------------------------------------
    // Checks what the minimum image size will be for the requested mip levels
    // if the request is too large, a adjusted minimum size will be returns
    //------------------------------------------------------------------------
    CMP_INT nMinSize = CMP_CalcMinMipSize(MipSetIn.m_nHeight, MipSetIn.m_nWidth, 10);

    //-------------------------------------------------------------
    // now that the minimum size is known, generate the miplevels
    // users can set any requested minumum size to use. The correct
    // miplevels will be set acordingly.
    //-------------------------------------------------------------
    CMP_GenerateMIPLevels(&MipSetIn, nMinSize);
}

//==========================
// Set Compression Options
//==========================
KernalOptions   kernel_options;
memset(&kernel_options, 0, sizeof(KernalOptions));

kernel_options.format   = destFormat;  // Set the format to process
kernel_options.fquality = fQuality;    // Set the quality of the result
kernel_options.threads  = 0;           // Auto setting

//---------------------------------------------------------------
// Setup a results buffer for the processed file,
// the content will be set after the source texture is processed
// in the call to CMP_ConvertMipTexture()
//---------------------------------------------------------------
CMP_MipSet MipSetCmp;
memset(&MipSetCmp, 0, sizeof(CMP_MipSet));

//================================================
// Compress the texture using Compressonator Lib
//================================================
cmp_status = CMP_ProcessTexture(&MipSetIn, &MipSetCmp, kernel_options,
→CompressionCallback);
if (cmp_status != CMP_OK) {
    CMP_FreeMipSet(&MipSetIn);
    std::printf("Compression returned an error %d\n", cmp_status);
    return cmp_status;
}
```

```
//---------------------------------------------------------------
// Save the result into a DDS file
//---------------------------------------------------------------
cmp_status = CMP_SaveTexture(pszDestFile, &MipSetCmp);
CMP_FreeMipSet(&MipSetIn);
CMP_FreeMipSet(&MipSetCmp);
```

Example projects have been provided here with:

- framework_example1 demonstrates simple SDK API usage by generating mipmap levels as shown above.

- framework_example2 demonstrates how to use the SDK API compression format, using a quality setting and a HPC pipeline framework.

- framework_exmaple3 demonstrates how to use the block level encoding SDK API.

These examples are also distributed through Compressonator Framework installer in the release page.

1.2.12 Using the Pipeline API Interfaces

These interfaces are designed to setup a specific data processing pipeline for:

- CPU generated code (1) using standard compilers such Visual Studio, GCC, Clang, . . .

- Vectorized CPU generated code (2) using SPMD (Single Process Multi Data) compilers such as ISPC compiler and libs.

- GPU Kernels using OpenCL, DirectX or Vulkan compilers.

To distinguish code used for (1) & (2) Compressonator uses the notation CPU & HPC respectively. The GPU setting is reserved for future release.

Compressonator framework sets up the path using Encoder Settings options for processing source data at a block level using the Compressonator Core. if the pipeline fails to setup then the data processing will default to the CPU (1) process.

To use the block level encoders, you must first create a specific encoder to access its block functions, this is done by calling CMP_CreateBlockEncoder (), by passing in a void reference pointer for the codec you want to create. This API requires you to specify which codec format type you are creating using a CMP_EncoderSetting structure format setting. An optional parameter is provided for setting the quality of the encoded blocks.

```
void *BC7encoder;
BC7encoder = NULL;
CMP_EncoderSetting encodeSettings;

encodeSettings.format = CMP_FORMAT_BC7;
encodeSettings.quality= 1.0f;

CMP_ERROR status = CMP_CreateBlockEncoder(&BC7block_encoder, encodeSettings);
```

If the create is successful the call will return CMP_OK else it will return a CMP_ERROR value. Once you have the reference pointer you can call the block encode CMP_CompressBlock () passing in the reference pointer and two block level buffers, one for the source and one for the compressed output.

```
status = CMP_CompressBlock(&BC7encoder, (void*)sourceBlock,(void*)compressBlock);
```

For users who want to use a fixed source buffer and destination buffer and have the codec process any specified block offset to them, a call to CMP_CompressBlockXY () is provided. In order for the call to process the correct buffer

offset, the original source buffer sizes must be provided during the codec create by specifying the size of the image using the EncoderSetting width and height parameters for the CMP_CreateBlockEncoder call.

```
encodeSettings.width  = SourceBufferWidth;
encodeSettings.height = SourceBufferHeight;
```

Once this is set the user can call CMP_CompressBlockXY which has a reference to the codec pointer and block locations x for column, y for the height and the fixed source and destination buffer pointers.

```
CMP_CompressBlockXY(&BC7encoder, x, y, (void*) sourceBufferData, (void*)
→compBufferData);
```

Both the source and destination buffers must be a correctly sized buffers for the encoders to use.

Once the processing is done the codec reference pointers can be removes from memory by calling CMP_DestroyBlockEncoder passing in the codec reference pointer.

```
CMP_DestroyBlockEncoder(&BC7encoder);
```

1.3 Compressonator SDK

Compressonator SDK's supported codecs includes BC1-BC7/DXTC, ETC1, ETC2, ASTC, ATC, ATI1N, ATI2N.

1.3.1 Error Codes

All Compressonator API calls return a int success 0 (CMP_OK) or error value > 0 (CMP_ERR) for a more detailed and up to date list look at the file Compressonator.h enum CMP_ERROR values.

```
CMP_OK = 0,                             // Ok.
CMP_ABORTED,                            // The conversion was aborted.
CMP_ERR_INVALID_SOURCE_TEXTURE,         // The source texture is invalid.
CMP_ERR_INVALID_DEST_TEXTURE,           // The destination texture is invalid.
CMP_ERR_UNSUPPORTED_SOURCE_FORMAT,      // The source format is not a supported format.
CMP_ERR_UNSUPPORTED_DEST_FORMAT,        // The destination format is not a supported
→format.
CMP_ERR_UNSUPPORTED_GPU_ASTC_DECODE,    // The gpu hardware is not supported.
CMP_ERR_UNSUPPORTED_GPU_BASIS_DECODE,   // The gpu hardware is not supported.
CMP_ERR_SIZE_MISMATCH,                  // The source and destination texture sizes do
→not match.
CMP_ERR_UNABLE_TO_INIT_CODEC,           // Compressonator was unable to initialize the
→codec needed for conversion.
CMP_ERR_UNABLE_TO_INIT_DECOMPRESSLIB,   // GPU_Decode Lib was unable to initialize the
→codec needed for decompression .
CMP_ERR_UNABLE_TO_INIT_COMPUTELIB,      // Compute Lib was unable to initialize the
→codec needed for compression.
CMP_ERR_CMP_DESTINATION,                // Error in compressing destination texture
CMP_ERR_MEM_ALLOC_FOR_MIPSET,           // Memory Error: allocating MIPSet compression
→level data buffer
CMP_ERR_UNKNOWN_DESTINATION_FORMAT,     // The destination Codec Type is unknown! In
→SDK refer to GetCodecType()
CMP_ERR_FAILED_HOST_SETUP,              // Failed to setup Host for processing
CMP_ERR_PLUGIN_FILE_NOT_FOUND,          // The required plugin library was not found
CMP_ERR_UNABLE_TO_LOAD_FILE,            // The requested file was not loaded
CMP_ERR_UNABLE_TO_CREATE_ENCODER,       // Request to create an encoder failed
```

(continues on next page)

```
CMP_ERR_UNABLE_TO_LOAD_ENCODER,        // Unable to load an encode library
CMP_ERR_NOSHADER_CODE_DEFINED,         // No shader code is available for the␣
→requested framework
CMP_ERR_GPU_DOESNOT_SUPPORT_COMPUTE,   // The GPU device selected does not support␣
→compute
CMP_ERR_GENERIC                        // An unknown error occurred.
```

1.3.2 Convert Texture

The main API call for both compression and decompression as well as texture conversion:

```
/// Converts the source texture to the destination texture
/// This can be compression, decompression or converting between two uncompressed␣
→formats.
/// \param[in] pSourceTexture A pointer to the source texture.
/// \param[in] pDestTexture A pointer to the destination texture.
/// \param[in] pOptions A pointer to the compression options – can be NULL.
/// \param[in] pFeedbackProc A pointer to the feedback function – can be NULL.
/// \param[in] pUser1 User data to pass to the feedback function.
/// \param[in] pUser2 User data to pass to the feedback function.
/// \return    CMP_OK if successful, otherwise the error code.
CMP_ERROR CMP_API CMP_ConvertTexture(CMP_Texture* pSourceTexture, CMP_Texture*␣
→pDestTexture, const CMP_CompressOptions* pOptions,
                                     CMP_Feedback_Proc pFeedbackProc, CMP_DWORD_PTR␣
→pUser1, CMP_DWORD_PTR pUser2);
```

1.3.3 Example Usage of Compressonator API

You will need to include a header file and a lib file: **Compressonator.h** and **Compressonator_MD.lib**

and a simple usage is shown as below:

```
//=========================
// Load Source Texture
//=========================
CMP_Texture srcTexture;
// note that LoadDDSFile function is a utils function to initialize the source CMP_
→Texture
// you can also initialize the source CMP_Texture the same way as initialize␣
→destination CMP_Texture
if (!LoadDDSFile(pszSourceFile, srcTexture))
{
    std::printf("Error loading source file!\n");
    return 0;
}


//==================================
// Initialize Compressed Destination
//==================================
CMP_Texture destTexture;
destTexture.dwSize      = sizeof(destTexture);
destTexture.dwWidth     = srcTexture.dwWidth;
destTexture.dwHeight    = srcTexture.dwHeight;
destTexture.dwPitch     = 0;
```

```cpp
destTexture.format     = destFormat;
destTexture.dwDataSize = CMP_CalculateBufferSize(&destTexture);
destTexture.pData = (CMP_BYTE*)malloc(destTexture.dwDataSize);

//=========================
// Set Compression Options
//=========================
CMP_CompressOptions options = {0};
options.dwSize       = sizeof(options);
options.fquality     = fQuality;
options.dwnumThreads = 8;

//=========================
// Compress Texture
//=========================
CMP_ERROR   cmp_status;
cmp_status = CMP_ConvertTexture(&srcTexture, &destTexture, &options, &
→CompressionCallback, NULL, NULL);
if (cmp_status != CMP_OK)
{
    free(srcTexture.pData);
    free(destTexture.pData);
    std::printf("Compression returned an error %d\n", cmp_status);
    return cmp_status;
}

//=========================
// Save Compressed Testure
//=========================
if (cmp_status == CMP_OK)
    SaveDDSFile(pszDestFile, destTexture);

free(srcTexture.pData);
free(destTexture.pData);
```

Example projects have been provided here with:

- sdk_example1 demonstrates simple SDK API usage as shown above.

- sdk_example2 demonstrates how to use the SDK API in multihreaded environment.

- sdk_exmaple3 demonstrates how to use the block level SDK API.

These examples are also distributed through Compressonator SDK installer in the release page.

1.4 Texture Compression and Decompression

For more details see Bibliography Reference (1)

1.4.1 BC1 Block (S3TC/DXT1)

BC1 block consists of two base colors c0 and c1 and an index table (bitmap).

The index table, however, has a two-bit entry, since BC1 allows for 2 additional colors, c2 and c3 obtained by blending of the base colors. All together c0, c1, c2 and c3 could be treated as a local palette for a compressed block. The base

colors are stored in RGB565 format, i.e. 5 bits for red and blue channels and 6 bit for green channel, resulting in 4bpp compression level.

There are two types of BC1 blocks: the first one that does not support transparency and the second one, that does.

1.4.2 BC2 Block (DXT2/DXT3)

The BC1 format can manage 24-bit RGB textures, but is unsuitable for 32-bit RGBA8888 textures. The BC2 block occupies 128 bit, twice the BC1 size. Therefore, compression level is 8bpp. One half of the BC2 is reserved for alpha values with 4-bit precision, the other one is just a BC1 for storing RGB data

1.4.3 BC3 Block (DXT4/DXT5)

The BC3 block, likewise BC2, consists of two 64-bit parts: one for the alpha data and one for the color data. Color part repeats the BC1 layout as well, but the alpha part is stored in the compressed form. Alpha compression is very similar to the DXT1 except for the number of the channels; there are two endpoints with 8-bit precision and the table of 3-bit indexes allowing to choose one of the eight values of a local palette.

1.4.4 BC4 Block (ATI1/3Dc+)

The BC4 block (Figure 9) is just an alpha part of the BC3 block. It is used for 1-channel textures, for example a height map or a specular map. Decoded values are associated with the red channel.

1.4.5 BC5 Block (ATI2/3Dc)

The 3Dc format was originally developed by ATI specially for the normal map compression, as the DXT1 format did not provide the required quality for such data. Normal map contains information about the direction of normal vector for every texel, which allows one to compute lighting with high level of detail and without increasing the geometry complexity.

1.4.6 BC6H

The BC6H format is designed to compress textures with high dynamic range (HDR). Only RGB images without alpha are supported. The format uses 128-bit blocks, resulting in 8bpp compression level. Depending on the block type, a compressed block has a different set of fields and a different size of each field. This allows choosing the best encoding on the per block basis. This flexibility greatly reduces compression artifacts, but strongly complicates the compression procedure. The number of block types has increased to 14 for BC6H and to 8 for BC7. Unlike BC1, block type is set explicitly in the first bits of compressed block. Block type is also referred to as the block mode.

1.4.7 BC7

Improves quality be adding new formats that improve the endpoint precision and storing up to three pairs of endpoints. The format uses 128-bit blocks, resulting in 8bpp compression level.

1.5 Mesh Optimization

Mesh optimization is only supported on Windows platform. Reference CMP_MeshOptimizer lib for implementation details.

1.6 Mesh Compression

As of v4.2 glTF draco mesh compression is no longer supported. Reference CMP_Mesh lib for alternate implementations.

Mesh compression/decompression provided in Compressonator by adding the following lines:

```
//=========================================
// Mesh Compression and Decompression
//=========================================
std::string       src_file = "source.gltf";      //input source glTF file
std::string       dst_file = "destination.gltf"; //output destination glTF file
std::string       err;                            //error messages
tinygltf2::Model    model;
tinygltf2::TinyGLTF loader;
tinygltf2::TinyGLTF saver;

bool perform_mesh_compression = true;  //flag to turn on/off compression
bool is_src_file_draco        = false; //flag to indicate source file is compressed␣
↪or not-
                                        //can be replaced with helper function␣
↪provided below to check for glTF compressed file

bool ret = loader.LoadASCIIFromFile(&model, &err, src_file, perform_mesh_compression);
if (ret)
    printf("read success");
else
    printf("read fail: %s", err);

err.clear();

CMP_CompressOptions CompressOptions;
// it is recommended to use only default settings, other settings may result in␣
↪corrupt in resource like texture.
CompressOptions.iCmpLevel   = CMP_MESH_COMP_LEVEL;    //setting: compression level␣
↪(range 0-10: higher mean more compressed) - default 7
CompressOptions.iPosBits    = CMP_MESH_POS_BITS;      //setting: quantization bits␣
↪for position - default 14
CompressOptions.iTexCBits   = CMP_MESH_TEXC_BITS;     //setting: quantization bits␣
↪for texture coordinates - default 12
CompressOptions.iNormalBits = CMP_MESH_NORMAL_BITS;   //setting: quantization bits␣
↪for normal - default 10
CompressOptions.iGenericBits = CMP_MESH_GENERIC_BITS; //setting: quantization bits␣
↪for generic - default 8

ret = saver.WriteGltfSceneToFile(&model, &err, dst_file, CompressOptions, is_src_file_
↪draco, perform_mesh_compression);

if (ret)
    printf("write success");
else
    printf("write fail: %s", err);
//=========================================
// end of Mesh Compression and Decompression
//=========================================
```

1.6.1 Helper function

```cpp
//Utility function to check for glTF draco compressed file
bool isGLTFCompressedFile(std::string filename)
{
    nlohmann::json j3;
    std::ifstream  f(filename);
    if (!f)
    {
        return false;
    }

    f >> j3;

    auto extrequired = j3["extensionsRequired"];

    for (int i = 0; i < extrequired.size(); i++)
    {
        std::string extname = extrequired[i].get<std::string>();
        if (extname.find("KHR_draco_mesh_compression") != string::npos)
        {
            return true;
        }
    }

    auto extused = j3["extensionsUsed"];

    for (int j = 0; j < extused.size(); j++)
    {
        std::string extnameused = extused[j].get<std::string>();
        if (extnameused.find("KHR_draco_mesh_compression") != string::npos)
        {
            return true;
        }
    }

    return false;
}
```

CHAPTER 2

Command Line Tool

2.1 Command Line Options

Usage CompressonatorCLI.exe [options] SourceFile DestFile

Mip Map Options:	
-GenGPUMipMaps	When encoding with GPU this flag will enable mip map level generation using GPU HW
-mipsize <size>	The size in pixels used to determine how many mip levels to generate
-miplevels <Level>	Sets Mips Level for output, (mipSize overides this option): default is 1
-nomipmap	Turns off Mipmap generation
-UseSRGBFrames	When encoding with GPU, GL_FRAMEBUFFER_SRGB will be enabled else use GL_FRAMEBUFFER
-FilterGamma <value>	Set a gamma correction value that will be applied after mipmap generation

Com- pres- sion Options	
-fd <for- mat>	Specifies the destination texture format to use
- DecodeWith	GPU based decompression using OpenGL,DirectX or Vulkan Default is OpenGL, UseGPUDecom- press is implied when this option is set
-decomp <file- name>	If the destination file is compressed optionally decompress it to the specified file. Note the destination must be compatible with the sources format,decompress formats are typically set to ARGB_8888 or ARGB_32F
- doswizzle	Swizzle the source images Red and Blue channels
- EncodeWith	Compression with CPU, HPC, OCL, DXC, GPU. Default is CPU. GPU will use GL Compress Exten- sions OCL & DXC is only available on Windows Version
- UseGPUDecompress	By default decompression is done using CPU, when set OpenGL will be used by default, this can be changed to DirectX or Vulkan using DecodeWith setting
- UseMangledFileName	Turns on name mangling, meaning processed files will have codec information appended to the end of the file name. Useful if you want to process multiple files with the same file name but different extensions.
-ff <ext>,…,<ext>	File filters used for selecting a subset of files in a directory folder for processing. The subset will contain only files that match any of the extensions given. Supported <ext> are any of the following combinations: DDS,KTX,TGA,EXR,PNG,BMP,HDR,JPG,TIFF,PPM,BRLG
-fx <ext>	Specifies the file extension to use for output files. Supported <ext> are any of the following values: DDS,KTX,TGA,EXR,PNG,BMP,HDR,JPG,TIFF,PPM,BRLG
- InExposure <value>	Set exposure tonemap property for float to non-float compression
- InDefog <value>	Set defog tonemap property for float to non-float compression
- InKneeLow <value>	Set knee low tonemap property for float to non-float compression
- InKneeHigh <value>	Set knee high tonemap property for float to non-float compression
-Gamma <value>	Set gamma tonemap property for float to non-float compression

Channel Formats	
ARGB_8888	ARGB format with 8-bit fixed channels
ARGB_16F	ARGB format with 16-bit floating-point channels
ARGB_32F	ARGB format with 32-bit floating-point channels

Compression Formats	
ATC_RGB	Compressed RGB format
ATC_RGBA_Explicit	ARGB format with explicit alpha
ATC_RGBA_Interpolated	ARGB format with interpolated alpha
ATI1N	Single component compression format using the same technique as DXT5 alpha. Four bits per pixel

Table 1 – continued from previous page

Compression Formats	
ATI2N	Two component compression format using the same technique as DXT5 alpha. Designed for compressio
ATI2N_XY	Two component compression format using the same technique as DXT5 alpha. The same as ATI2N but v
ATI2N_DXT5	An ATI2N like format using DXT5. Intended for use on GPUs that do not natively support ATI2N. Eight
BC1	Four component opaque (or 1-bit alpha) compressed texture format. Four bit per pixel
BC2	Four component compressed texture format with explicit alpha. Eight bits per pixel
BC3	Four component compressed texture format with interpolated alpha. Eight bits per pixel
BC4	Single component (red channel)compressed texture format
BC4_S	Signed Channel compression using BC4 format
BC5	Two component (reg and green channels) compressed format
BC5_S	Signed Channel compression using BC5 format
BC6H	High-Dynamic Range compression format
BC7	High-quality compression of RGB and RGBA data
DXT1	An opaque (or 1-bit alpha) DXTC compressed texture format. Four bits per pixel
DXT3	DXTC compressed texture format with explicit alpha. Eight bits per pixel
DXT5	DXTC compressed texture format with interpolated alpha. Eight bits per pixel
DXT5_xGBR	DXT5 with the red component swizzled into the alpha channel Eight bits per pixel
DXT5_RxBG	Swizzled DXT5 format with the green component swizzled into the alpha channel. Eight bits per pixel
DXT5_RBxG	Swizzled DXT5 format with the green component swizzled into the alpha channel & the blue componen
DXT5_xRBG	Swizzled DXT5 format with the green component swizzled into the alpha channel & the red component
DXT5_RGxB	Swizzled DXT5 format with the blue component swizzled into the alpha channel. Eight bits per pixel
DXT5_xGxR	Two-component swizzled DXT5 format with the red component swizzled into the alpha channel & the g
ETC_RGB	Ericsson Texture Compression - Compressed RGB format.
ETC2_RGB	Ericsson Texture Compression 2 - RGB format
ETC2_RGBA	RGB with 8 bit alpha
ETC2_RGBA1	RGB with 1 bit alpha
BRLG	Lossless compression using Brotli-G

Codec Options	Reference developer SDK documentation for range of values
-AlphaRestrict <value>	This setting is a quality tuning setting for BC7 which may be necessary for some textures
-AlphaThreshold <value>	The alpha threshold to use when compressing to DXT1 & BC1 with DXT1UseAlpha Texels with an alpha value less than the threshold are treated as transparent value is in the range of 0 to 255, default is 128
-Analysis <image1> <image2>	Generate analysis metric like SSIM, PSNR values between 2 images with same size. Analysis_Result.xml file will be generated.
-ColourRestrict <value>	This setting is a quality tuning setting for BC7 which may be necessary for convenience in some applications
-CompressionSpeed <value>	The trade-off between compression speed & quality This setting is not used in BC6H and BC7
-diff_image <image1> <image2>	Generate difference between 2 images with same size A .bmp file will be generated. Please use compressonator GUI to increase the contrast to view the diff pixels.
-DXT1UseAlpha <value>	Encode single-bit alpha data. Only valid when compressing to DXT1 & BC1
-imageprops <image>	Print image properties of image files specifies.
-log	Logs process information to a process_results.txt file containing file info, performance data,SSIM,PSNR and MSE.
-logcsv	Logs process information to a process_results.csv file containing file info, performance data,SSIM,PSNR and MSE.
-log <file-name>	Logs process information to a user defined text file
-logcsv <file-name>	Logs process information to a user defined csv file
-ModeMask <value>	Mode to set BC7 to encode blocks using any of 8 different block modes in order to obtain the highest quality
-NumThreads <value>	Number of threads to initialize for BC6H and BC7 encoding (Max up to 128). Default set to 0 (Auto)
-Performance <value>	Sets performance of encoding for BC7
-PageSize <value>	Page size, in bytes, to use for Brotli-G compression
-Quality <value>	Sets quality of encoding for BC7
-RefineSteps <value>	Adds extra steps in encoding for BC1 to improve quality over performance. Step values are 1 and 2.
-Signed <value>	Used for BC6H only, Default BC6H format disables use of a sign bit in the 16-bit floating point channels, with a value set to 1 BC6H format will use a sign bit
-UseChannelWeighting <value>	Use channel weightings
-WeightR <value>	The weighting of the Red or X Channel
-WeightG <value>	The weighting of the Green or Y Channel
-WeightB <value>	The weighting of the Blue or Z Channel

Chapter 2. Command Line Tool

Output Options	
-noprogress	Disables showing of compression progress messages
-performance	Shows various performance stats
-silent	Disable print messages

2.1.1 Example Compression

CompressonatorCLI.exe -fd BC7 -EncodeWith HPC image.bmp result.dds CompressonatorCLI.exe -fd BC7 image.bmp result.dds CompressonatorCLI.exe -fd BC7 -NumTheads 16 image.bmp result.dds CompressonatorCLI.exe -fd BC6H image.exr result.dds

2.1.2 Example Compression using GPU

CompressonatorCLI.exe -fd BC1 -EncodeWith GPU image.bmp result.dds CompressonatorCLI.exe -fd BC1 -EncodeWith OCL image.bmp result.dds CompressonatorCLI.exe -fd BC1 -EncodeWith DXC image.bmp result.dds

2.1.3 Example Decompression from compressed image using CPU

CompressonatorCLI.exe result.dds image.bmp

2.1.4 Compression Followed by Decompression

(Useful for qualitative analysis)

CompressonatorCLI.exe -fd BC7 image.bmp result.bmp

2.1.5 GPU Based Decompression

compressonatorCLI.exe -DecodeWith OpenGL result.dds image.bmp

2.1.6 Mesh Optimization

(Only supports glTF and obj files)

The following uses default settings that optimizes vertices with cache size = 16, overdraw with ACMR Threshold = 1.05 and vertices fetch.

compressonatorcli.exe -meshopt source.gltf dest.gltf

compressonatorcli.exe -meshopt source.obj dest.obj

Specifies settings:

compressonatorcli.exe -meshopt -optVCacheSize 32 -optOverdrawACMRThres 1.03 -optVFetch 0 source.gltf dest.gltf

CLI mesh optimization has the following settings:

-optVCacheSize <value>	optimize vertices with hardware cache size in the value specified Default is enabled with cache size = 16
-optVCacheFIFOSize <value>	optimize vertices with hardware FIFO cache size in the value specified Default is disabled
-optOverdrawACMRThres <value>	optimize overdraw with ACMR (average cache miss ratio) threshold value specified (value Ranges1-3) default is enabled with ACMR value = 1.05 (i.e. 5% worse)
-optVFetch <boolean value>	optimize vertices fetch . boolean value 0 - disabled, 1-enabled. -default is enabled.
-simplifyMeshLOD <value>	simplify mesh using LOD (Level of Details) value specified. (value range 1- no limit as it allows users to simplify the mesh until the level they desired. Higher level means less triangles drawn, less details.)

2.1.7 Test Analysis Logging Features and File Filters

(Windows OS only)

CLI will generate an output "process_results.txt" when -log is added to the compression command line options, users can change the default log file using the command -logfile, the log captures details of the source and destination files along with statistical data on performance and quality.

Example:

```
C:\>CompressonatorCLI -log -fd BC7 .\images\ruby.png ruby_bc7.dds
```

Generates a "process_results.txt" file with content:

```
CompressonatorCLI Performance Log v1.0

Source       : .\images\ruby.png, Height 416, Width 576, Size 0.936 MBytes
Destination  : ruby_bc7.dds
Using        : CPU
Quality      : 0.05
Processed to : BC7        with  1 iteration(s) in 1.422 seconds
MSE          : 0.78
PSNR         : 49.2
SSIM         : 0.9978
Total time   : 1.432 seconds


--------------
```

Multiple processes will append results to this file with a dash line separator. The option is valid only for compressing images and not for 3D models or image transcoding.

In addition to the -log and -logfile two command-line options are avilable to output analysis data into comma-separated file format. use -logcsv or -logcsvfile to generate a .csv file suitable to use in any application that supports viewing these files in a table as shown in this sample:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	CompressonatorCLI Performance Log v1.1															
2																
3	Negative values are errors in measurement															
4	For images with no errors PSNR=255 MSE=0 and SSIM=1.0															
5																
6	Source	Height	Width	LinearSize(MB)	Destination	ProcessedTo	Iteration	Duration(s)	Using	Quality	KPerf(ms)	MTx/s	MSE	PSNR	SSIM	TotalTime(s)
7	.\images\ruby.bmp	416	576	0.958	ruby_bc1.dd	BC1	0	0.158	HPC	1	10.016	1.597	5.09	41.1	0.99	0.166
8	.\images\ruby.bmp	416	576	0.958	ruby_bc1.dd	BC1	0	0.601	OCL	1	0.5	31.97	5.1	41.1	0.99	0.615

The CLI also support processing image files from a folder, without the need to specify a file name. Using a file filter, specific files types can also be selected for compression as needed.

Examples:

```
C:\>CompressonatorCLI -fd BC7 .\images .\results
```

Processes all image file with BC7 Compression into results folder

```
C:\>CompressonatorCLI -fd BC7 -ff BMP,PNG,EXR .\images .\results
```

Processes only images with extension bmp, png and exr. Notice that BC7 compression is been applied to HDR images, this is an automatic Adaptive Channel Format feature (ACF) that transcodes the image half float channels to byte prior to processing.

2.1.8 CSV File Update to Support Automation

An error code field is added to log the state of a processed image when using the command-line application option "-logcsv".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	CompressonatorCLI Performance Log v1.2																
2																	
3	Negative values are errors in measurement Sets ErrCode > 0 else 0 for none																
4	For images with no errors MSE= 0 PSNR=255 and SSIM= 1.0																
5	Transcoded images MSE= 0 PSNR=255 and SSIM=-2.0																
6	No image data generated MSE=-1 PSNR= -1 and SSIM=-1.0 with ErrCode set																
7																	
8	Source	Height	Width	LinearSize(MB)	Destination	ProcessedTo	Iteration	Duration(s)	Using	Quality	KPerf(ms)	MTx/s	MSE	PSNR	SSIM	TotalTime	ErrCode
9	./images/ruby.bmp	416	576	0.958	./results/ruby_ATI2N_DXT5.ktx2	ATI2N_DXT5	1	0	CPU	0.05	0	0	-1	-1	-1	0	1
10	./images/ruby.bmp	416	576	0.958	./results/ruby_BC1.ktx2	BC1	1	0.07	CPU	0.05	0	0	4.9259	41.2	0.9915	0.070	0
11																	

The error code will be 0 for processed images, else a value is set to indicate any errors encountered while the image was processed.

For a list of the most recent codes look for AnalysisErrorCodeType in the sdk file cmp_compressonatorlib/common.h

GUI Tool

3.1 Getting Started with Compressonator GUI

This guide provides detailed information on the Compressonator application. It lists the requirement needed for running the application and helps in installation, getting started with the tool, using the sample projects and finding specific topics of interest.

3.1.1 Compressonator GUI features

- The GUI interacts with Compressonator SDK for texture compression and bit format conversions; it can compress a wide range of compression formats including ATC, ATInN, BCn, ETCn, DXTn, swizzle DXTn formats.

- Supports conversion of textures with 32 bit fixed and float formats.

- Process multiple compression, decompression and transcode of images with a single processing action.

- Allow multiple processing interactions for a single source image

- Inspect visually and analytically compression results.

- Uses a single image viewer that supports a large number of compressed and uncompressed image formats.

3.1.2 Application Requirements

- Application was tested on Windows 10 platform.

- For best performance, a multi core CPU system with the latest AMD Graphics card are required.

- For ease of use, a wheel based mouse is essential.

- Use of multiple monitors is optional, and can be beneficial when examining multiple image views at the same time. Installation

- To install the application, download the executable file from release page to your system. Then, double click the executable to start the installation process and follow the on-screen instructions.

- Run the application in Administrator mode, recommended that UAC be turned off.

3.1.3 Using Sample Projects

This section shows you how to get started with the application using the sample projects that come with the installation. Note that you can also start by creating a new project and add your image files for compression.

1. From the start menu or desktop shortcut run the application

2. You will see the following view
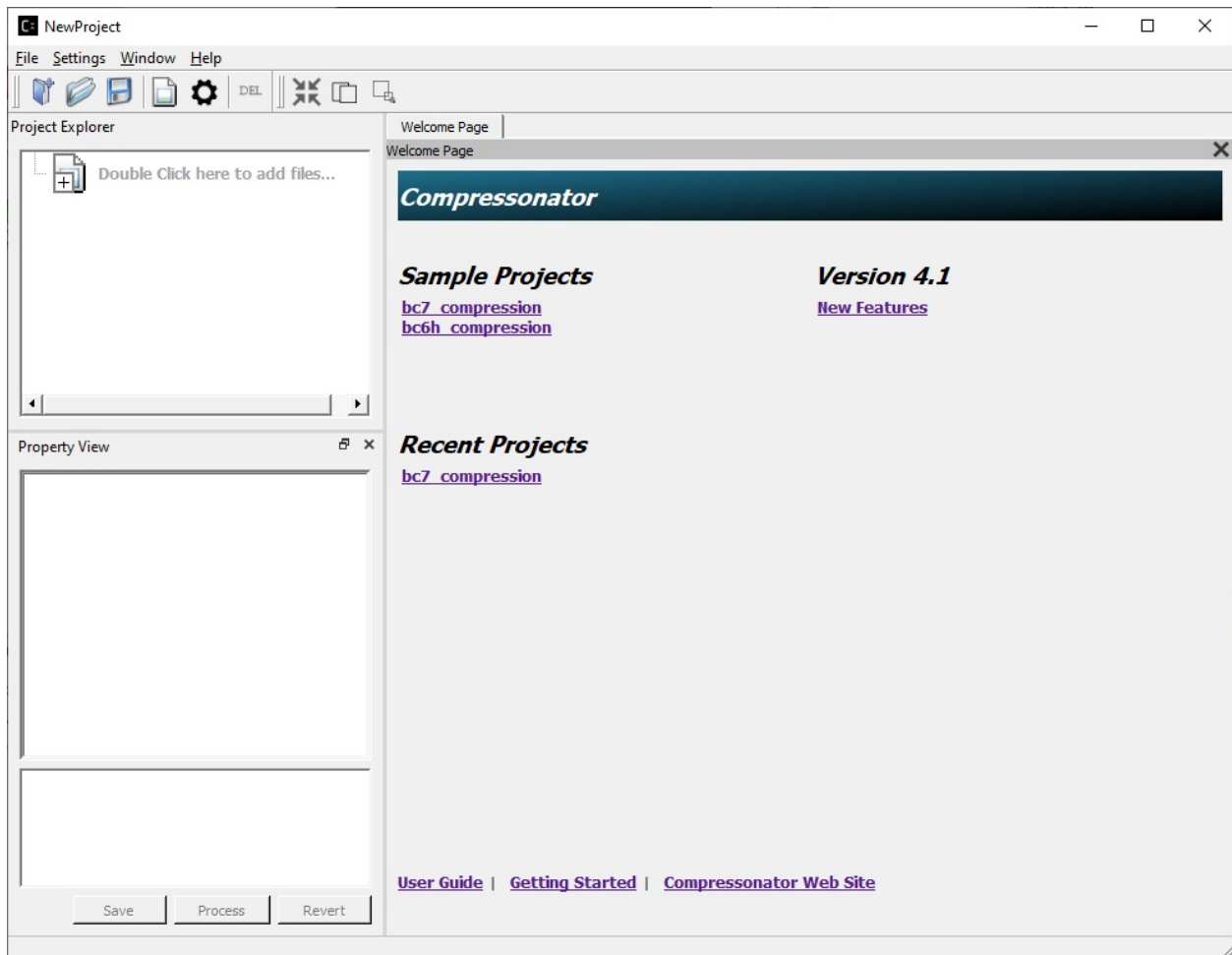
3. On the Welcome Page tab Window, click on "BC7_Compression", the Project Explorer will change and show some sample images and settings
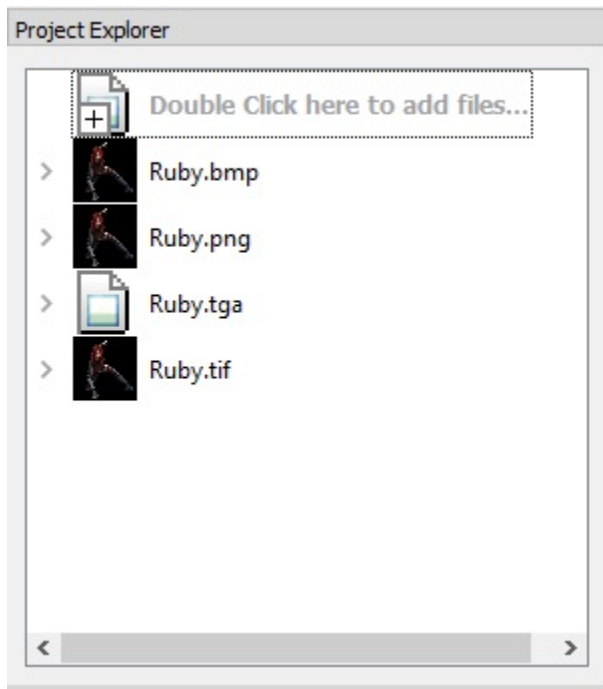
4. Click on Ruby.bmp file and you will see an Image View window show up on the right of the Project Explorer, tabbed with the Welcome Page window (as shown below).
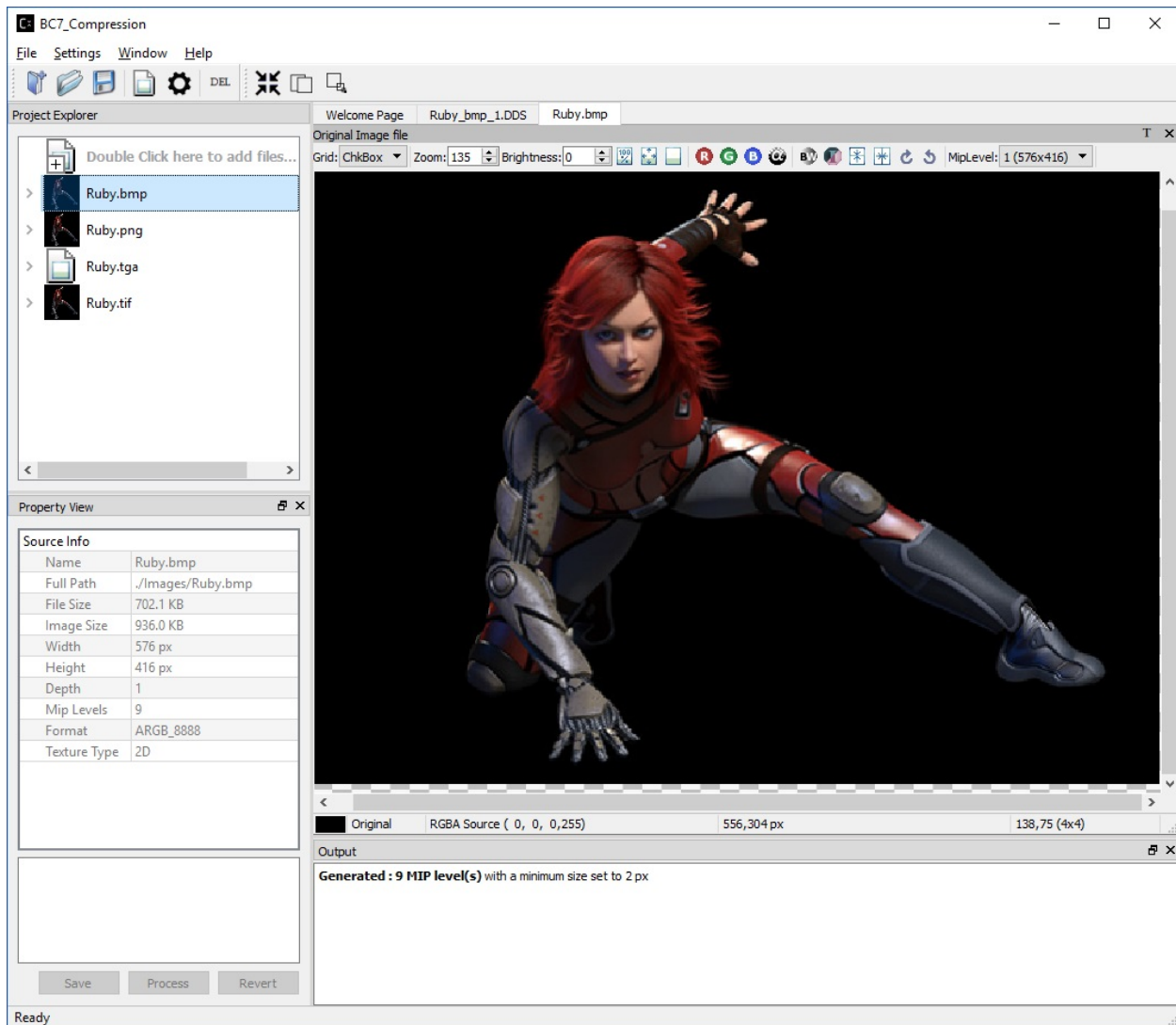
The Properties View will now display information on the selected image's location, various sizes, dimensions, etc.

5. Now click on the right arrow next to the Ruby.bmp.
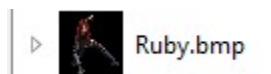
This expands the view and you will see a clickable "Add destination settings …" line and a BC7 pre-compressed destination sample Ruby_1

6. Click on Ruby_1, and notice that the Properties View changed (as shown below) to indicate what settings has been preset for Ruby_1
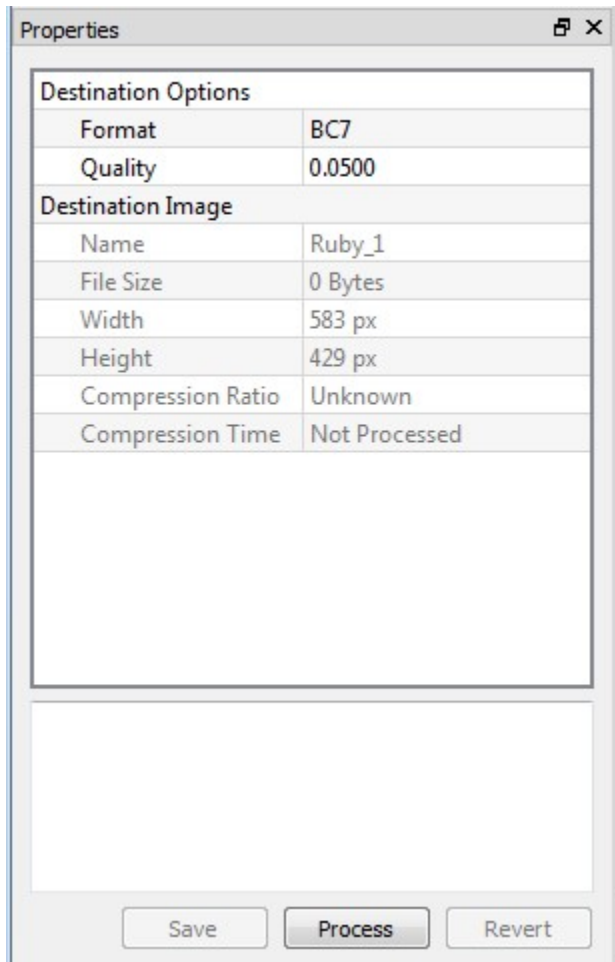
Ruby_1 is set to compress the original Ruby.bmp image using BC7 compression format, the expected quality of the resulting image is shown as default 0.05, this value ranges from 0 to 1. Note that lower quality value will have faster compression process with less amount of precision when compared to the original. On the other hand, higher quality value will slow down the compression process but produce better image quality.

7. To start the compression, click on "Process" button, a Progress windows and an Output window appear.

8. When the compression process done, the Project Explorer will change to indicate the status of the resulting compressed Ruby_1 image with a small green (succeeded) or red circle (failed), and the Output window will indicate additional information on the succeeded or failed compression process.

Output

```
Processing: Ruby_bmp_1
CMD Line: ./Images/Ruby.bmp  ./Images/Ruby_bmp_1.DDS  -fd BC7  -Quality 0.0500
Source Texture size = 1000428 Bytes, width = 583 px  height = 429 px
Destination Texture size = 252288 Bytes  Resulting compression ratio = 3.97:1
Processing destination
Compressed to BC7 with 1 iteration(s) in 0.398 seconds
DeCompressed to ARGB_8888 with 1 iteration(s) in 0.099 seconds
Total time taken (includes file I/O): 0.503 seconds
Done
====== Compress: 1 succeeded, 0 failed ======
```
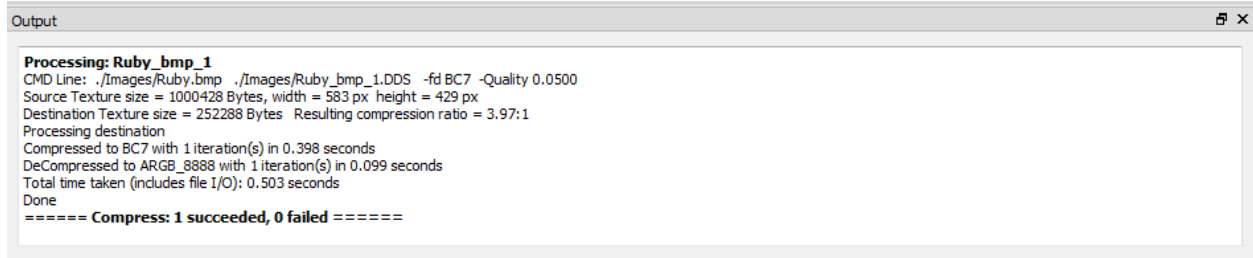
For this sample, we should see a green circle next to Ruby_1 (compression succeeded)

Ruby.bmp

Add destination settings...

Ruby_bmp_1
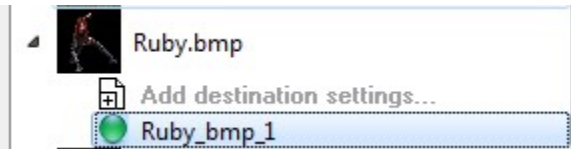
9. Now the Properties View will indicate the time it took to compress the image. To see the Compression Ratio, click on RUby_1 again, this will update the Compression Ratio which indicates how much the image was compressed compared to the original (typically 4x)

Properties

Destination Options	
Format	BC7
Quality	0.0500
Destination Image	
Name	Ruby_bmp_1
File Size	246.5 KB
Width	583 px
Height	429 px
Compression Ratio	3.96 to 1
Compression Time	0.503 Sec
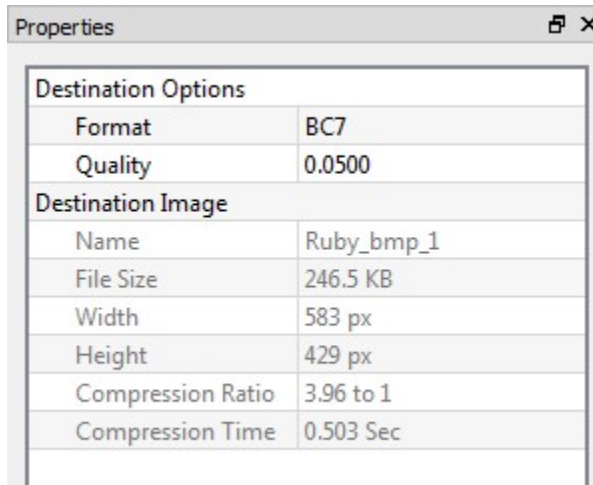
10. To see the resulting compressed image, single click on Ruby_1 and you will see the image as shown below.
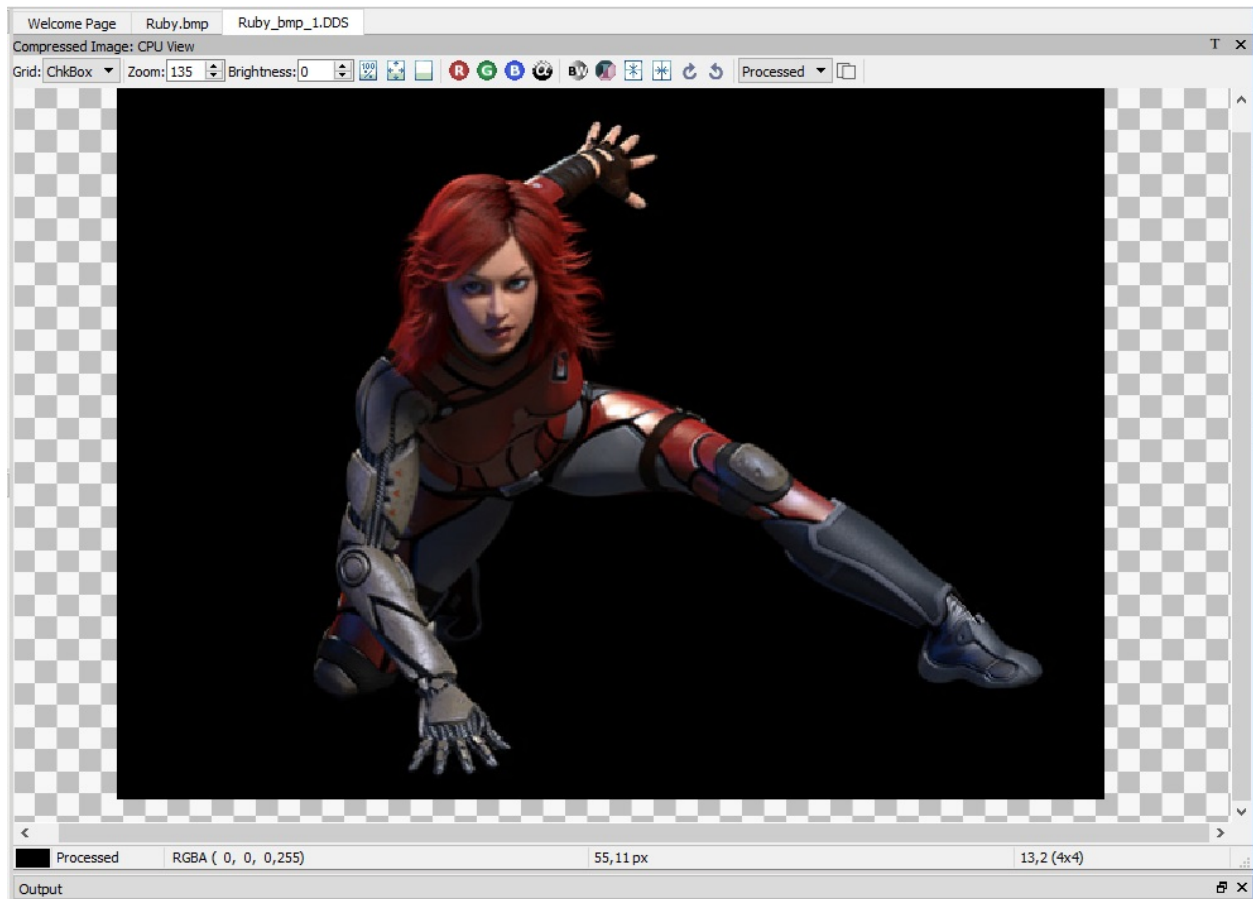
11. To view the difference between processed image (Ruby_1) and original image (Ruby.bmp), right click on Ruby_1 and select View Image Diff from the context menu
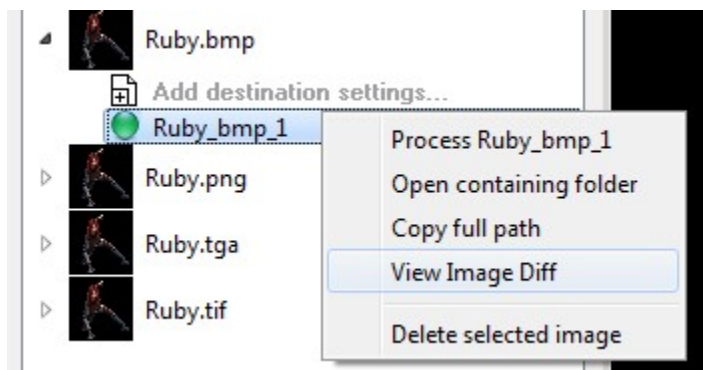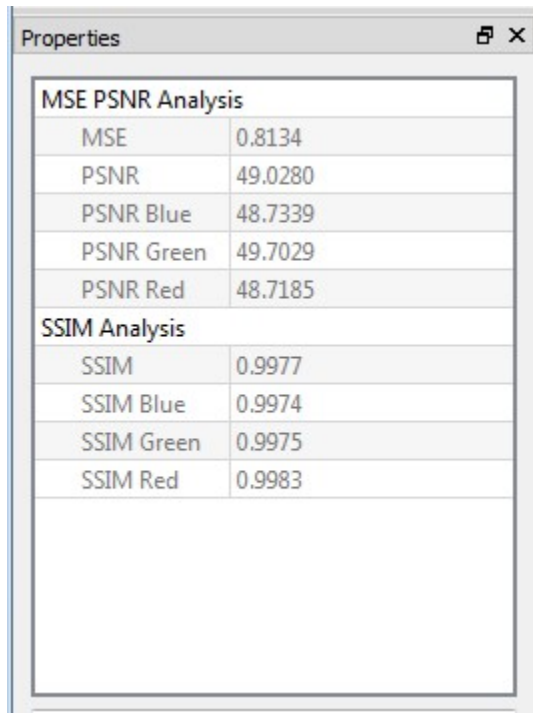
You will now see a comparison of the original image with the compressed image
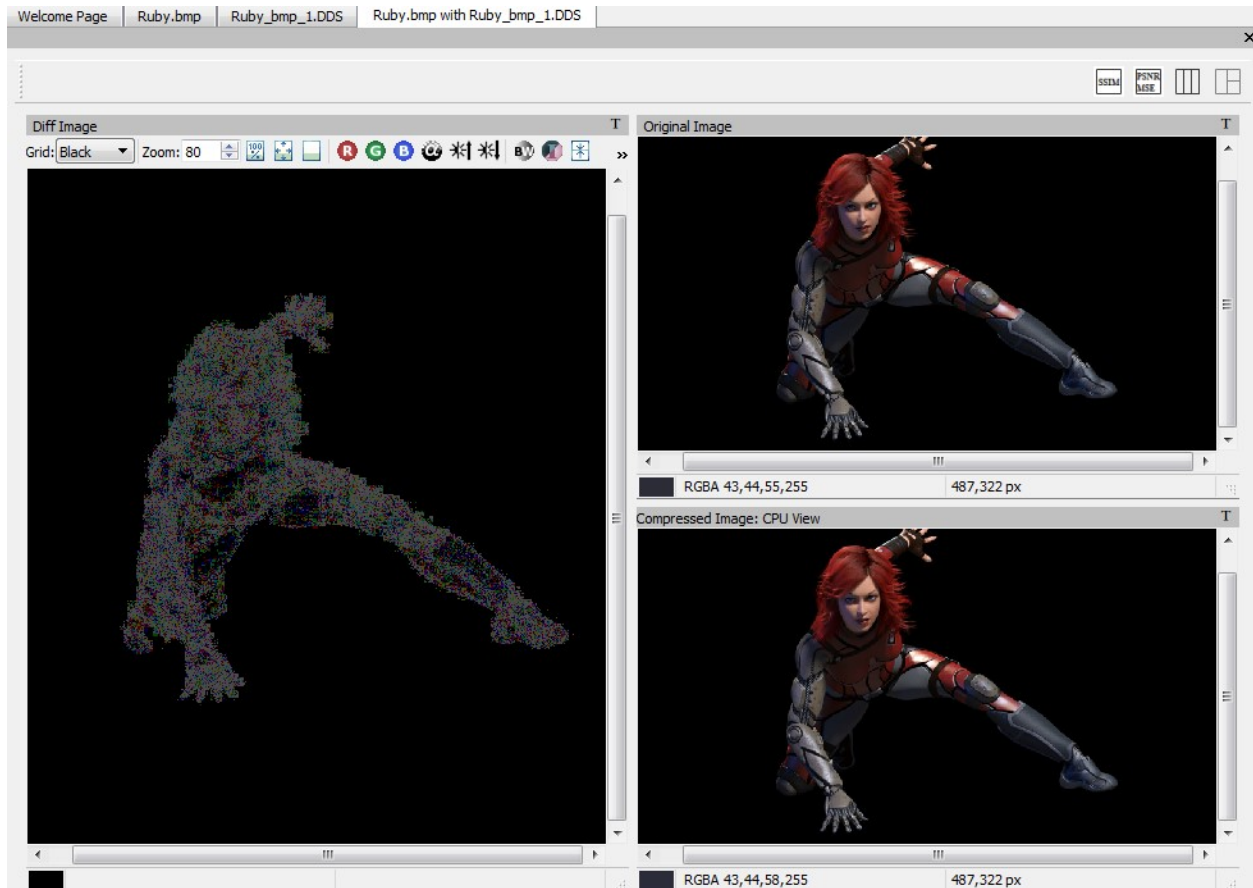
12. In addition, you can run analysis on the images that show various statistics such as MSE, PSNR and Similarity Indices (SSIM) by selecting

When analysis process completed, the statistics result will be shown on the Properties View:

3.1.4 Closing views

At any time you can close various views by selecting the close button (on tabs or windows)

3.1.5 Changing views

Click on Image View window tab titles or click on any image on Project Explorer.

3.1.6 Delete or Remove Image(s)

Select the image in Project Explorer and press DEL key. A message window will pop up as shown below.
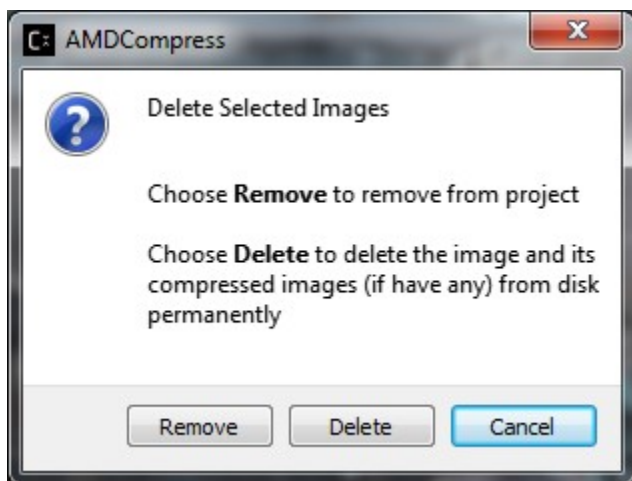
Remove Will remove the selected items from the project explorer view, keeping the files on disk.

Delete Will Remove and delete files on disk.
 Cancel Will return you back to the application (similar applies when selecting the red close button)

3.1.7 Closing and Saving the Project

Select File menu and then Exit or Click on the close button on the application window Since we have no changes to the settings or added any new images, the application will simply close when exit. If you made any changes to the sample project "BC7_Compression" the application will prompt to save the changes or discard them. When you select save the old settings for "BC7_Compression" will be overwritten.
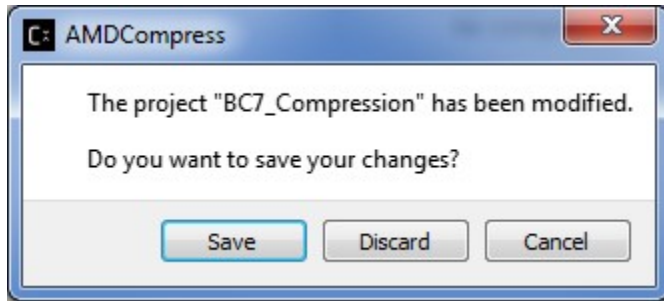
3.2 GUI User Guide

3.2.1 Introduction

The Compressonator GUI (Graphical User Interface) application, commonly shortened to just Compressonator GUI, is meant to be a user-friendly way to access all of Compressonator's many features. It allows users to do things like: texture compression & decompression, generating mipmaps, viewing a wide variety of images, image analysis, and more without needing to write any code or interface with a command line.

This page will go over the basic requirements and installation of Compressonator GUI.

System Requirements

Compressonator GUI is currently only available for the Windows operating system. It has primarily been tested on Windows 10 and Windows 11, though it may work with other versions as well.

For optimal performance, a multi-core CPU paired with the latest AMD GPU is strongly recommended.

Installation

Installation is pretty simple, download the latest "CompressonatorGUI" executable file from GPU Open and double-click to install like you would with any other Windows application.

3.2.2 The Main Window

Image View

The Image View is the main window in Compressonator GUI. It is where you can view a wide variety of source image formats, see the results of compression, inspect image analysis metrics, and much more.

To open an image in the Image View you must left-click on the image icon next to an item in the Project Explorer. Multiple images can be opened simultaneously and switched between by selecting the appropriate tab at the top of the Image View.

Viewing Images

The Image Viewer in combination with the Project Explorer allows users to view a large variety of image formats. Ref *Adding Addition Image Formats* on what formats are supported.

Note: In some cases viewing an image may take some time to display if the CPU is used to decompress a large compressed image item.

- To display an original image view, single mouse click on the item on the Project Explorer.

- To display a destination item image view double click mouse on the Item on the Project Explorer.
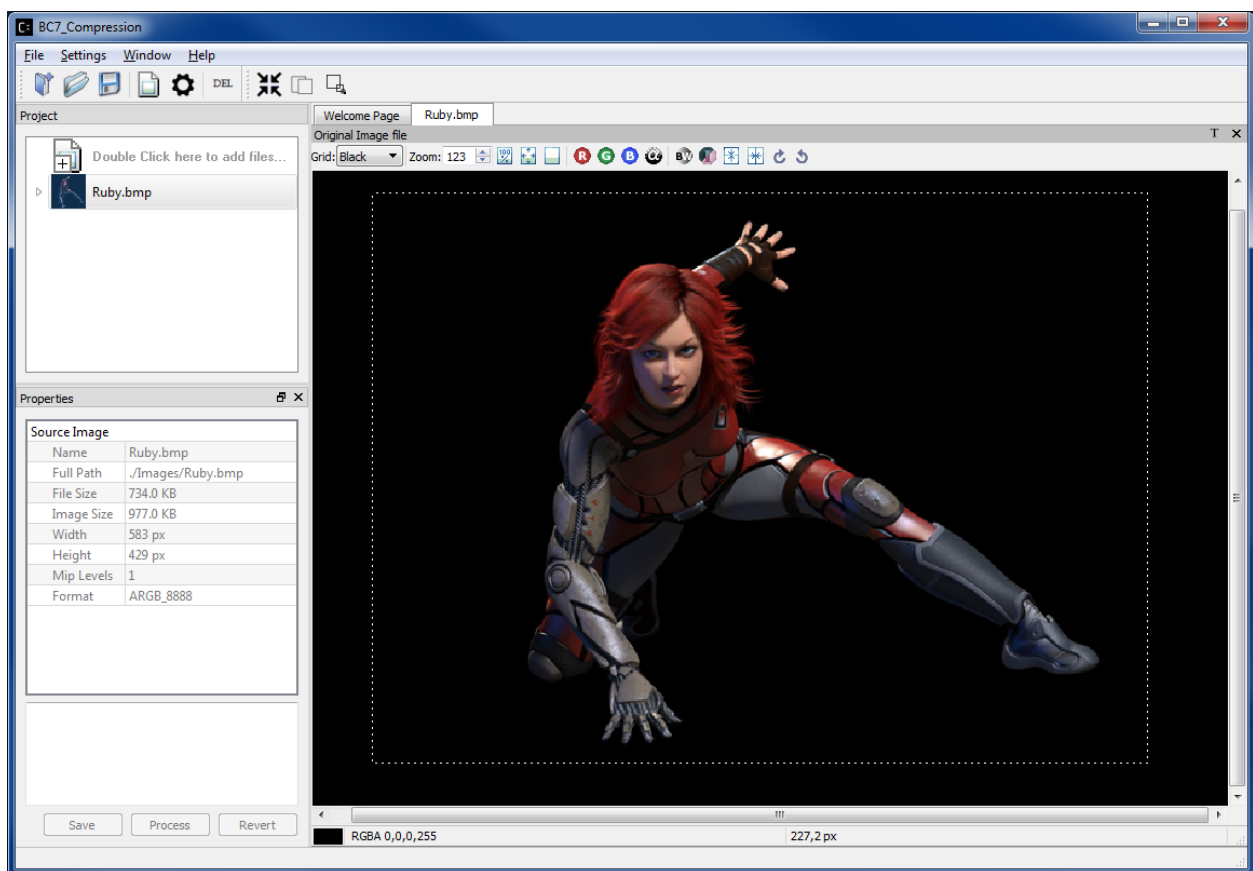
Image View of Ruby.bmp

To view a processed image (Project Explorer destination item with a green circle), double click on it (Ruby_bmp_1 in this example) and you will see another Image View tab window appear displaying the processed image.
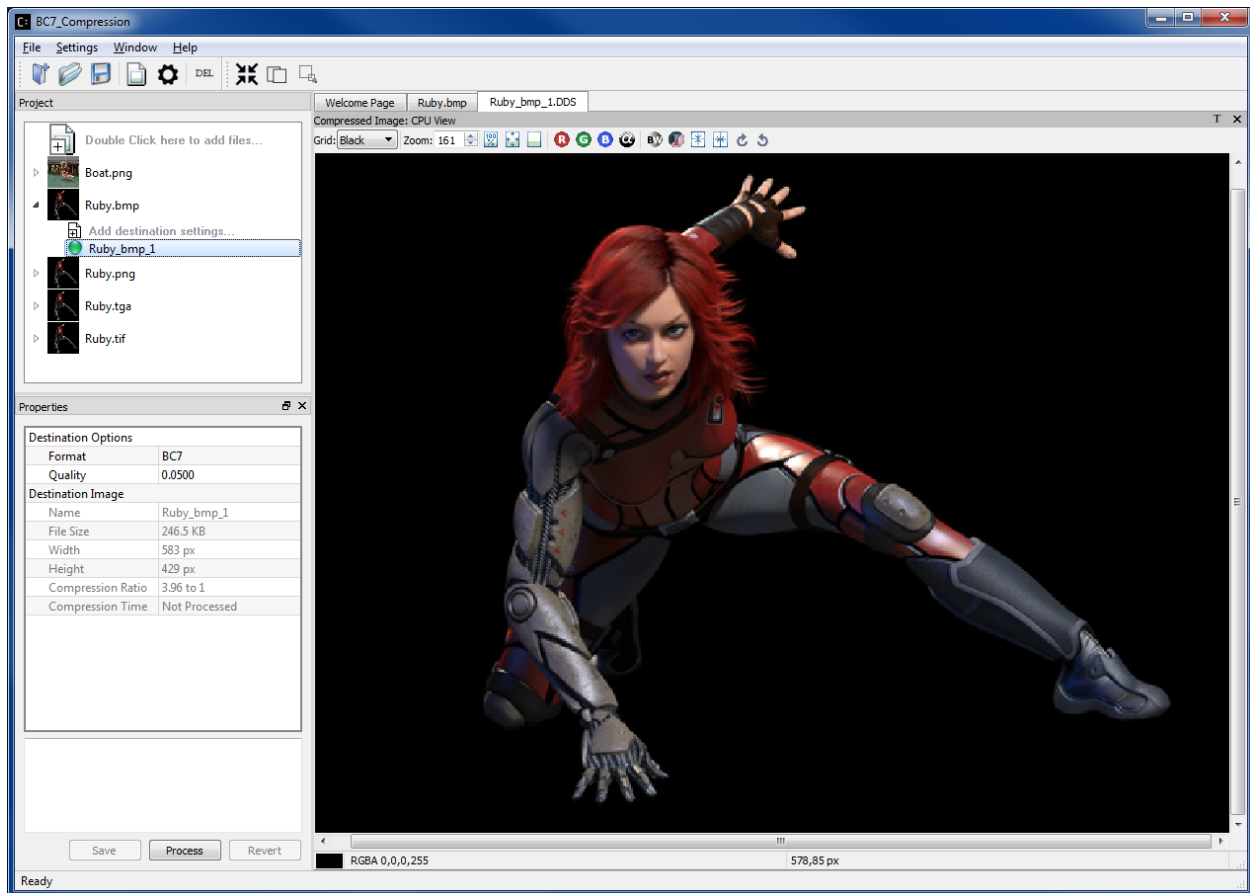
BC7 Compressed Image View of Ruby_bmp1.dds file

Users can capture viewed images to file using context menu "Save View as" or keyboard keys Ctrl C (displayed image), Alt C (original source image) to save to Windows Clipboard.
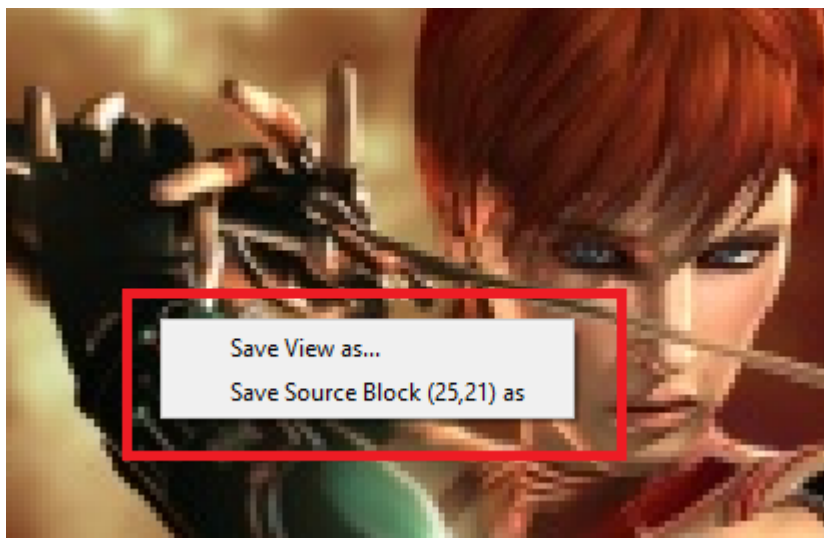
Image view context menu (enabled by right mouse button click)

Cursor positions in block increments is also displayed, users can now save any block to file using "Save Source Block ... as" where ... is the current cursor location translated to a block position.
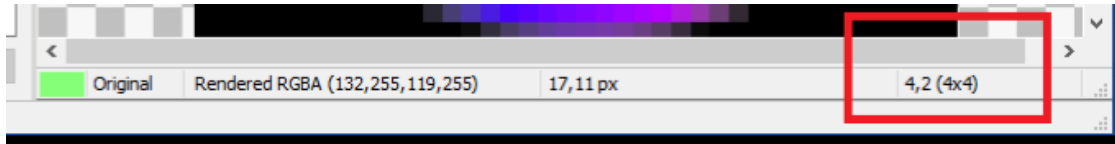
Image view status bar showing cursors block position

This feature is useful in capturing the source texture blocks that was used for generating compressed blocks that exhibit decompressed image artifacts or poor quality. Depending on the source, a 4x4 block image is saved to either BMP or EXR file formats, which can later be used for repeated analysis.

Using a Mouse Wheel for Zoom

Rotating the mouse wheel in or out will also zoom the image in increments of 10. To zoom at a faster rate (increments of 100) hold down the keyboard Ctrl key and rotate the mouse wheel. Zoom will be centered on the current mouse cursor location.

Using the Mouse to Move the Image in the View

To move the image around the view, place the mouse over the image and press down the left mouse button. While keeping it pressed, move the image around the view. To stop the move, release the left mouse button.

Note: you can only perform these operations when the mouse cursor is over the image.

Adding additional Image Formats

There are a number of default image file formats that are supported by the application

Format	Description
BMP	Windows Bitmap
PNG	Portable Network Graphics
EXR	High Dynamic Range Images
DDS	Direct Draw Surface files
KTX, KTX2	Khronos Texture Files
TGA	Targa Texture files
TIFF	Tagged Image File Format files

Support for EXR, TGA, DDS and KTX are linked into the main application and cannot be overwritten

Users can add additional file format by placing appropriate Qt Image plugins into \plugins\imageformats folder.

KTX2 Support Notes

KTX2 is supported by a DLL in the plugin folder AMD Compress\plugins\imageformats and at the root folder as ktx.dll it supports saving and loading multichannel images, BCn, and ETCn codecs.

The following custom image formats supported in Compressonator KTX is not available in KTX2 ATC_RGB,ATC_RGBA_Explicit,ATI1N,ATI2N,ATI2N_XY, DXT5_xGBR, ATI2N_DXT5, DXT5_xGBR,DXT5_RxBG ,DXT5_RBxG,DXT5_xRBG,DXT5_RGxB and DXT5_xGxR.

For ATI1N use BC4 and for ATI2N use BC5.

Additional support for universal textures and streaming can be added upon request. For specific KTX2 feature enablement, please file a request at https://github.com/GPUOpen-Tools/compressonator/issues

Image View switch between Original and Processed

This feature allows users to switch quickly between Original Image View and Processed Image View (Processed refers to Compressed Image View or Pixels Difference Image View) by simple key stokes (O or P) or from the Image View bar dropdown list. This allows users to have a visual comparison between Original and Processed images.

After *compressing image*, click on the green circle beside the compressed Image to generate the image view window as shown below:
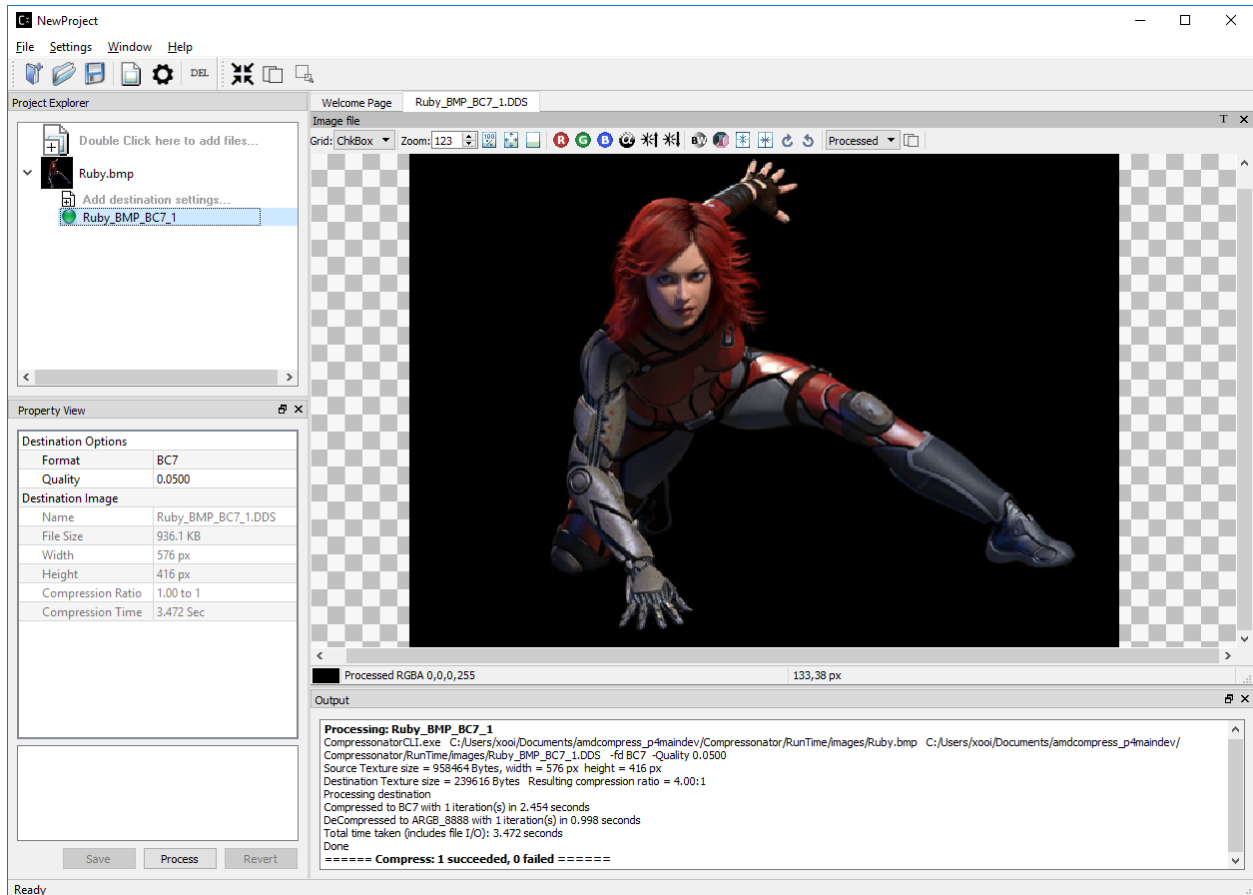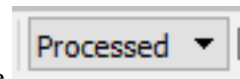
Click on the Image View window to set focus, then hit "Space" bar to switch between Original and Processed (In this case, Processed refers to Compressed Image) Image Views. You can also switch between Processed and and Original
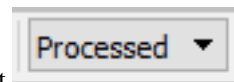
Views by select the related views from the combo drop down list
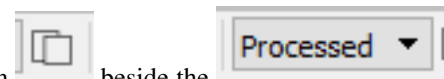
To change the Processed Image View to Image Diff View, click on the icon beside the
, observed that Processed View now shows Image Diff View (In this case, Processed refers to Image Diff), you can switch between Original and Image Diff View by hitting the "Space" bar. Note: You may need to increase the Image

Brightness by clicking on to view the Processed Image Diff.

To change the Processed Image View back to Compressed Image View, click on the same icon ⬜.

PSNR Image Quality Metric

When viewing the compressed result of an image some analysis metrics are calculated and can be viewed. One such metric is the PSNR. It can be seen in the status bar at the bottom of the image view and updates based on which mipmap level or cubemap face is currently being viewed.

The PSNR value can be used to numerically compare the quality of various types of compressed images and filter options applied to an image.

Toolbars

There are several types of tool bars. Depending on user preferences, some can be moved to a new location on the application and others can either be displayed or hidden by selecting the [T] button.

Note: The Toolbar view setting is reset when the application is restarted.

Application Toolbar

	Create a new project
	Open an existing project file (project files have the extension .cprj)
	Saves any changes to the current project
	Open image files to add to the Project Explorer
	Open an application settings dialog
DEL	Delete the current selected items on the Project Explorer
	Process all selected images (that might be compresssion, decompression, or transcoding) If no items are selected when the button is clicked, it will default to processing every item in the project.
	View image difference for a processed image (sub item) with the original image (branch item)
	Generate mipmaps for every selected image based on the specified minimum size.

Image View Toolbar
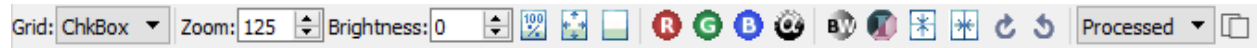
Grid: Black ▼	Grid: ChkBox ▼ Grid: Lines ▼ Grid: Points ▼ Select the background the image is displayed on (default black)
Zoom: 122 ⬍	Zoom into or out of the image using a specific scale (100 = original size) Min setting is 10 and Max is 9999
⬜	Displays the image at its original size
⬜	Sets the image display to fit the current view
⬜	Restore the original image view to default
R	Toggles the image's red channel on or off
G	Toggles the image's green channel on or off
B	Toggles the image's blue channel on or off
◉	Toggles the image's alpha channel on or off
B	Toggles display of a greyscale version of the image
◐	Toggle between original and inverted colours
❋	Mirrors the image horizontally
❋	Mirrors the image vertically
↻	Rotates the image clockwise by 90 Degrees
↺	Rotates the image counter clockwise by 90 Degrees

Additional Toolbar options will be displayed when viewing certain type of images, such as an image diff or images with multiple mipmap levels

❋	Increases the brightness of the image
❋	Decreases the brightness of the image
MipLevel: 0 (768x512) ▼	Selecting a mipmap level in the dropdown will switch the image view to display that specific mipmap level (index 0 is the original image).

View Image Diff Toolbar

	| Displays an Image Difference to the left of two stacked images. Original on top and Processed Image at the bottom.
	| Organizes three image views (Original Image, Image Difference and Processed Image) in a horizontal line
PSNR MSE	The Property View displays Peak Signal Noise Ratio and Mean Square Error for the processed image as compared to the original. **MSE** measures the cumulative squared error between the processed image and original image. (Value of 0 = Original image) **PSNR** measures the peak error in (dB) – (extremely high value = original image)
SSIM	Structural Similarity Index (**SSIM**) measures a perceived quality factor from a scale of 0 (= No similarity) to 1 (= Original image). | Each channel of the processed image is indexed with the original and then averaged to a single value.

Application Options

Various default application behaviors can be set using the Application Settings window. Select ⚙ from the application tool bar or from the Settings – Set Application Options menu
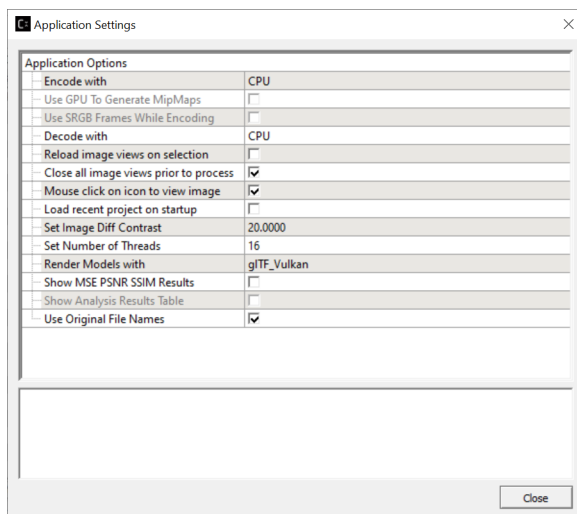
Application Settings Window

Encode with

For compressed images, this option selects how images are compressed either with CPU, HPC, or with GPU using shaders for OpenCL (OCL), DirectX Compute (DXC), or Graphics Hardware Extension (GPU). these options can be selected from the GUI using "Encode With" HPC runs codecs optimized for vector extensions and SPMD processing on CPU.

Note Only BC1 to BC7 format is supported with HPC Compress, if you choose another format under HPC Compress, they will be compressed with generalized CPU instructions See the latest release notes for details on what GPU codecs are available for encoding.

Graphics Hardware Extension (GPU) is set for BCn codecs.
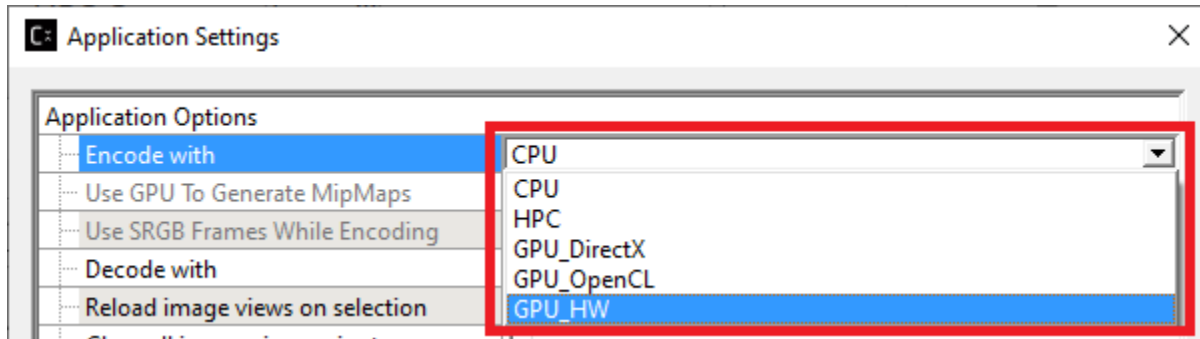
Decode with

This option is a drop-down combo list where users can choose to decode the image with CPU, GPU_OpenGL, GPU_DirectX, or GPU_Vulkan. It is used when users click to view an image on the Image View window.

Reload Image Views on Selection

This option when set will always close the current image view and open a new image view. This is useful when an image has been processed to say a new compression format and changed visually from when it was last viewed. By default, this is turned on (check-marked). If you turn this option off then the view will not be refreshed every time you click on viewing an image from Project Explorer. The advantage of switching this mode is that for large compressed images the image view takes considerable time to decompress and not necessary if the compressed file content has not changed.

Close all Image Views Prior to Process

This option when set will close all Image Views in the application, before processing selected image destination settings in the Project Explorer. This will free up system memory, to avoid out of memory issues when processing large files.

Mouse click on icon to view image

This option is checked by default. When checked, the application will load the image/model onto the Image View window when the user clicks on the icon next to the image file node in Project Explorer. When it is unchecked (off), the application will load the image/model onto the Image View window when the user clicks on the image filename or icon.

Load Recent Project on Startup

This option off by default will load the last project you worked on. This saves you time selecting it from the welcome page or the recent files list from the file menu.

Set Image Diff Contrast

Sets the contrast of pixels for image view diff, using 1.0 returns pixels to original diff contrast, min is 1 max is 200

Set Number of Threads

Sets the number of threads to use for texture compression, max is 128 threads distributed over multiple cores Default 0 sets auto-detection, where the total threads = number of processor cores, if auto-detection fails default = 8

When selecting this option users can also view what the host processor has available and is shown in square brackets as illustrated below:

Max number of processors [8]

The GUI application must be restarted in order for the new settings to take effect

Render Models with

Selects how to render 3D Model files using OpenGL for Obj and DirectX or Vulkan for GLTF files.

Show MSE PSNR SSIM Results

The output windows will display these values after processing a image, when all processing is done an average summary of all of the results will be displayed.

Show Analysis Results Table

Shows all Process Times, PSNR and SSIM results for compressed images in a table view.

Use Original File Names

Enabled by default, this option can be used to toggle whether generated destination file names should use the same name as the source. This only applies to the first destination for each source file. To prevent file overwriting, subsequent destination files will have the codec type and an index appended to the file name.

Property View

Properties such as file size and image format of a selected item on the Project Explorer are displayed on the Property View. Content of the Property View changes when different items are selected in the application.

Properties

The Properties View will display information on the selected image's location, various sizes, dimensions, etc.

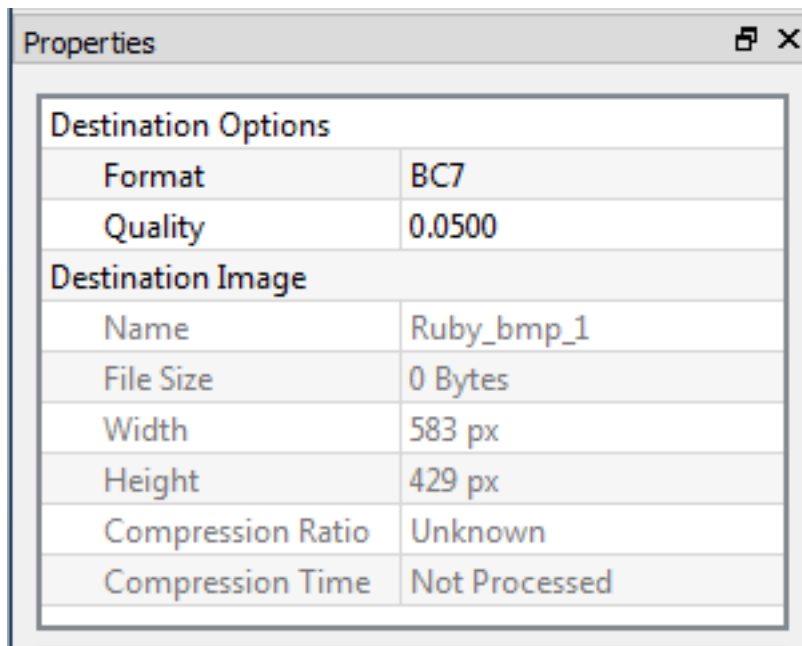| Properties	
Destination Options	
Format	BC7
Quality	0.0500
Destination Image	
Name	Ruby_bmp_1
File Size	0 Bytes
Width	583 px
Height	429 px
Compression Ratio	Unknown
Compression Time	Not Processed
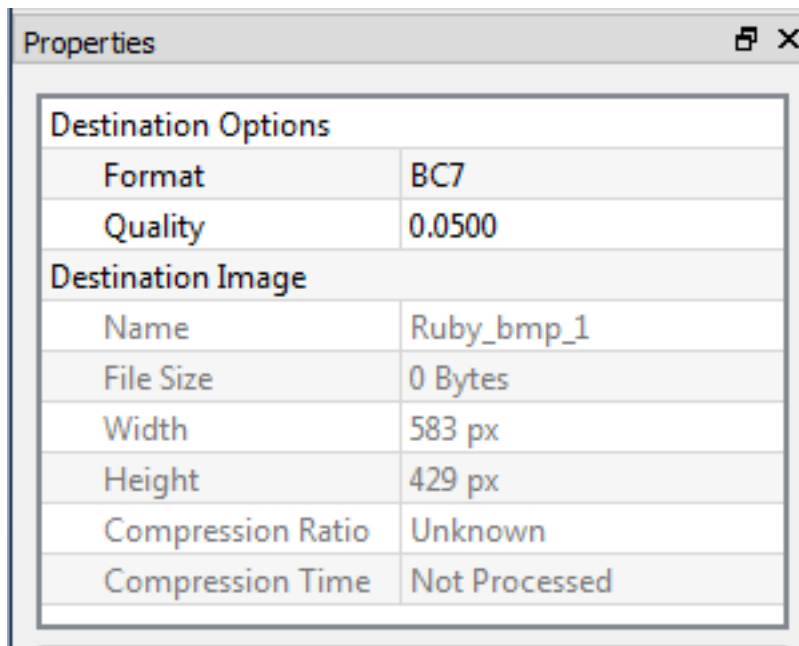
Property View Window

The Property View above shows that the selected image is set to compress the original Ruby.bmp image using BC7 compression format, the expected quality of the resulting image is shown as default 0.05, this value ranges from 0 to 1. Lower quality values will have faster processing time and less amount of precision when compared to the original image.

Warning: For some large images, setting quality values above 0.75, the time to process it may take several hours for only a marginal increase in overall quality when compared to the original image.

When a compression process is completed, the Property View will indicate the time it took to compress the image and the Compression Ratio. To see the Compression Ratio, click on compressed image in the Project Explorer, this will update the Compression Ratio which indicates how much the image was compressed compared to the original (typically 4x for BC7)

Properties	
Destination Options	
Format	BC7
Quality	0.0500
Destination Image	
Name	Boat_1
File Size	384.1 KB
Width	768 px
Height	512 px
Compression Ratio	4.00 to 1
Compression Time	1.041 Sec
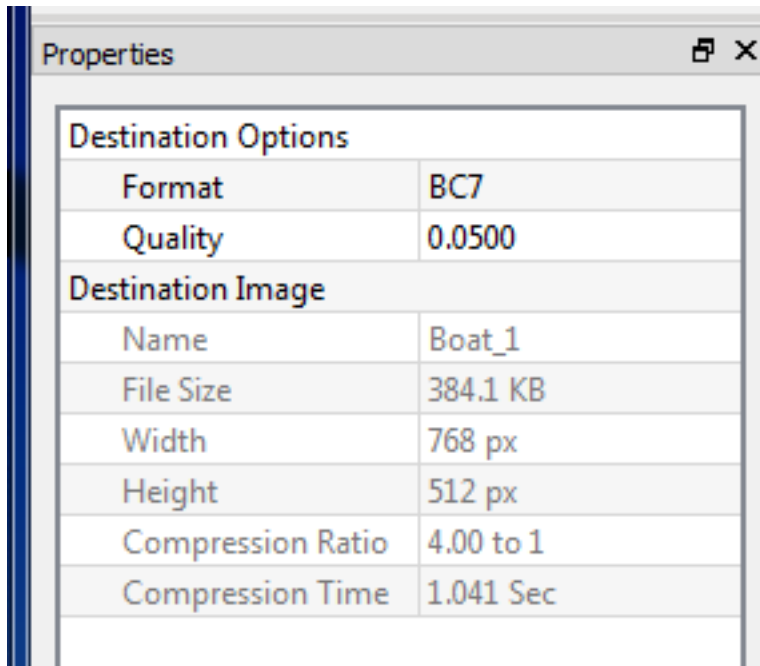
Property View Window showing Compression Ratio

Project Explorer

Images and models are added to the application using the Project Explorer.

2D texture image files are displayed in two levels, original source image is shown on the left (branch item) and destination image(s) (sub items) are shown on the right using a tree view outline.

3D model files (only support .obj and .gltf with .bin) are displayed in three levels, original model is shown on the left (branch item) and destination model is shown as 2nd level sub item while the mesh/textures within the model is shown as 3rd level item of the tree view.

Projects

The application uses a project based concept, where 2D texture images are added to the Project Explorer tree view as original image items in which settings are applied using a destination item. Each original 2D texture image item can have multiple destination items. A destination item can be set to generate a file with a specified format (compressed, decompressed or transcoded) and extension (DDS, KTX, BMP, etc.)

While for 3D model items, they are added to the Project Explorer tree view as original model items, in which multiple model destination settings can be added as 2nd level tree which create multiple resulted model items, the 3rd level destination setting which applied to the mesh/texture items within the model can only be added once per mesh/texture item.

Multiple destination items can be processed at the same time.

Projects can be loaded, created and saved to disk at any time.

Sample Projects

These samples can be accessed either from the Welcome Page or from the sample projects folder

---

Compressonator\Projects

BC7_Compression.pproject file demonstrating compression using BC7 on images with source extension formats of BMP, PNG, TGA and TIF	
BC6H_Compression.pproject file demonstrating compression using BC6H on a high dynamic range image (OpenEXR) file extension format of (EXR)	

Processing Ruby.bmp sample using BC7 Compression

1. On the Welcome Page tab window as shown in the view below, click on "BC7_Compression"
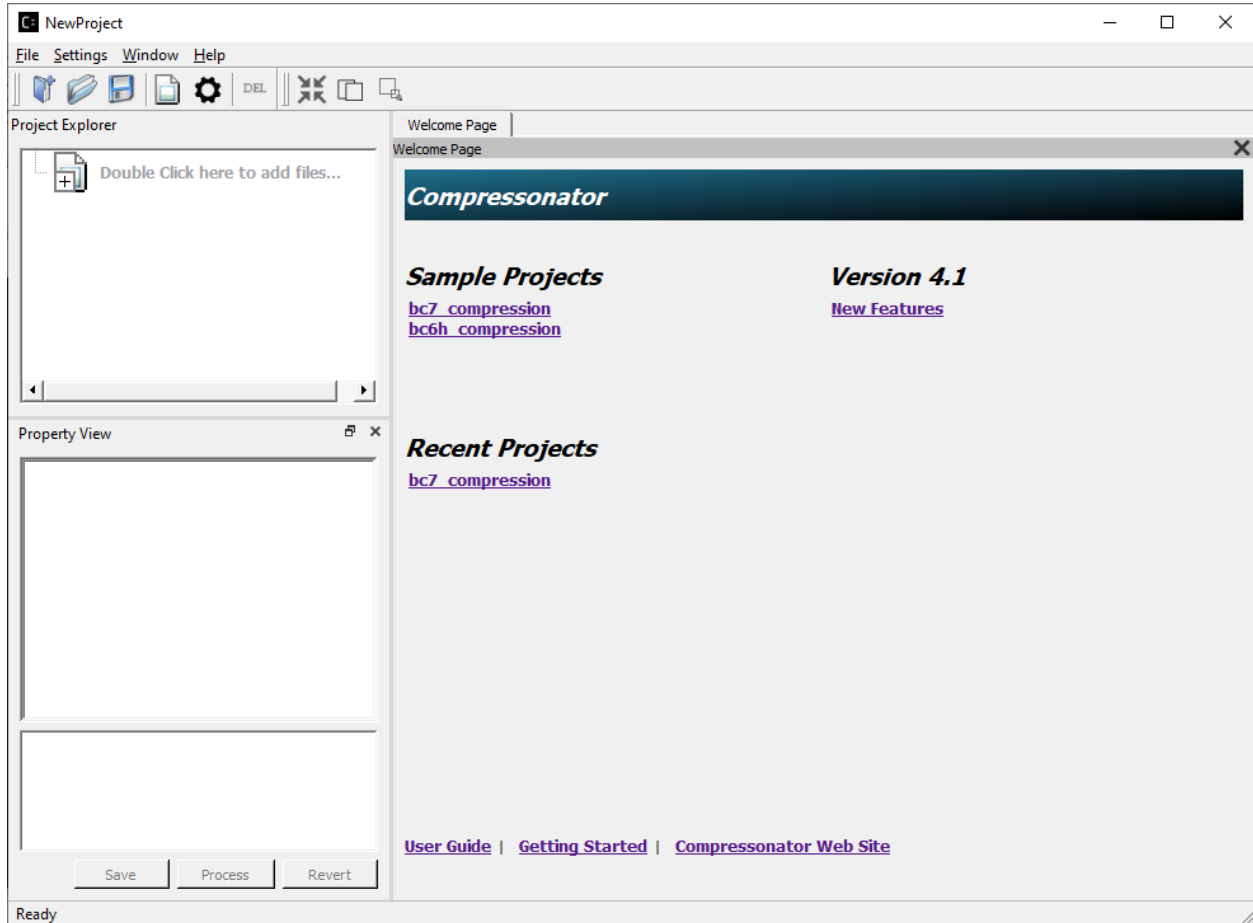
The Project Explorer will change and show some sample images and settings from the BC7 Compression sample project:
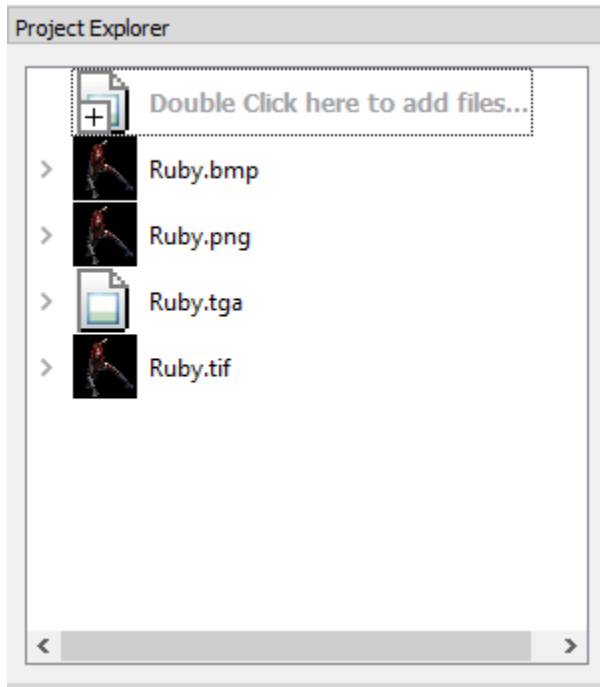
2. Select the image by clicking on the name (for example, Ruby.bmp), the Properties View will now display information on the selected image's location, various sizes, dimensions, etc.

3. Now click on the right arrow next to the Ruby.bmp.
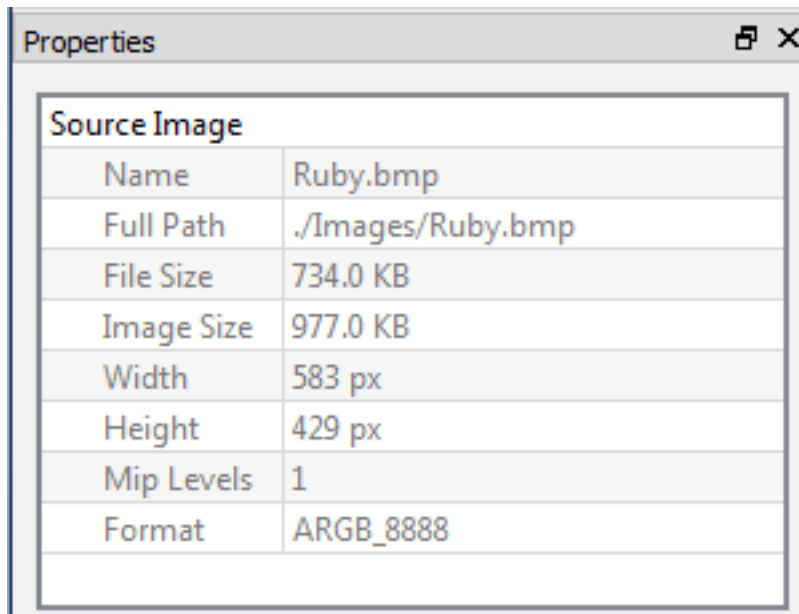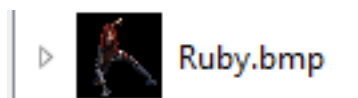
This expands the view and you will see a clickable "Add destination settings …" line and a BC7 pre-compressed destination sample Ruby_bmp_1.

Ruby.bmp

Add destination settings...
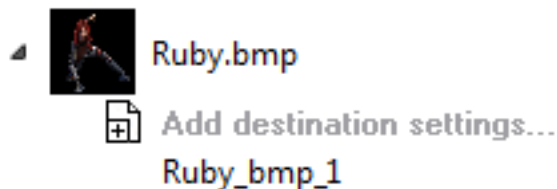
Ruby_bmp_1

4. Click on Ruby_bmp_1, and notice that the Property View changed (as shown below) to indicate what settings has been preset for Ruby_bmp_1

Note that Compression Ratio and Compression Time both show "Unknown" and "Not Processed". These values will be updated when the destination file is created during processing.

5. Click on the Process button located in the Properties View. Two new windows will open a Progress Window and a Message Output window. When processing is complete the progress window will close and the Output window will show a result.
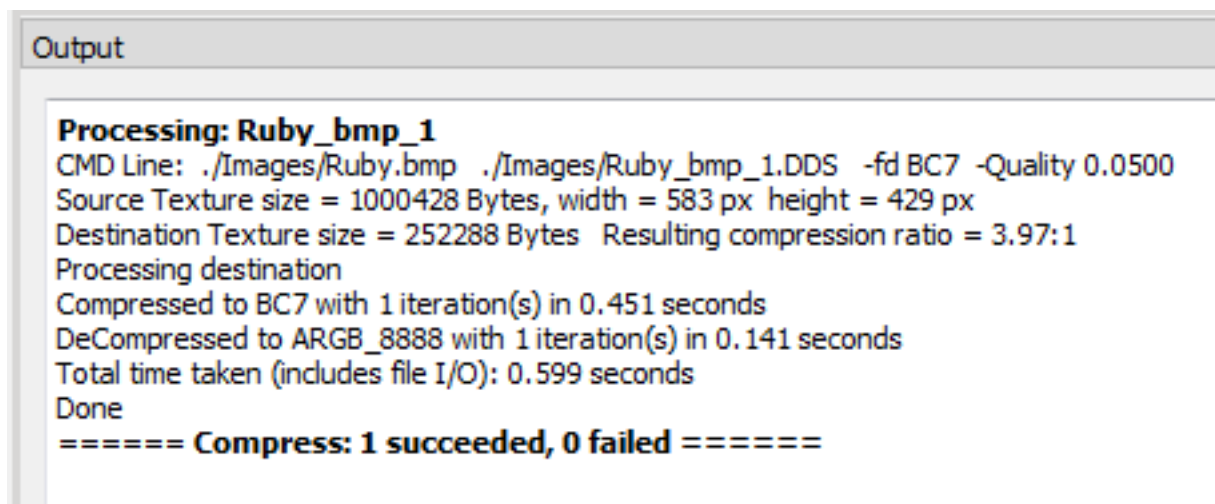
Notice also that there is a green circle next to Ruby_bmp_1, indicating that a compressed file has been created and the process was successful.
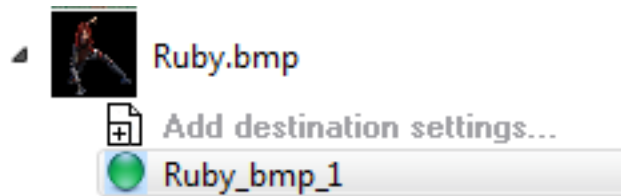
6. To view the resulting file, double click on Ruby_bmp_1
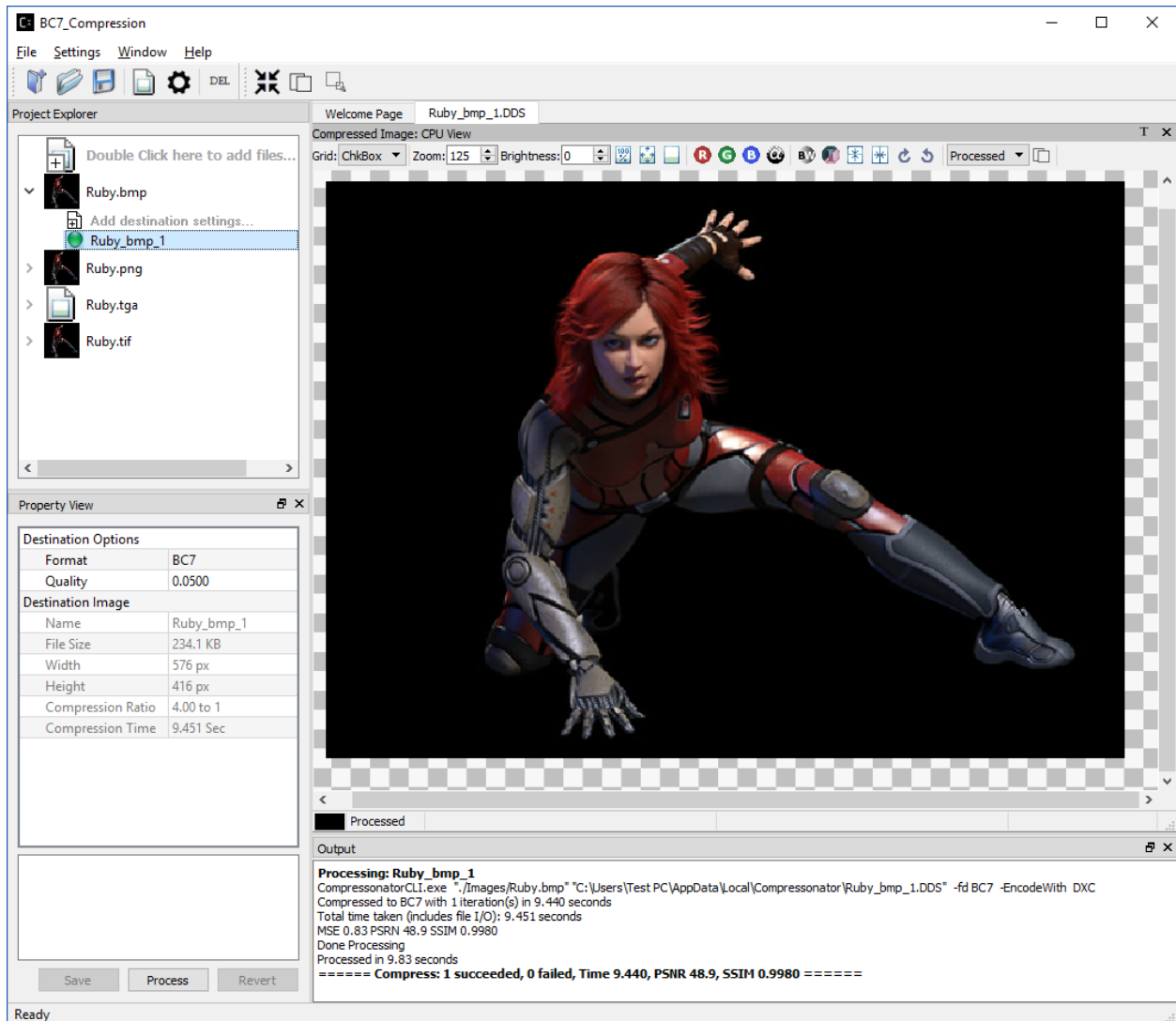
Add Destination Settings

To add new destination settings a for specific original image (branch item), expand its branch and select Add destination settings... by double clicking on it.
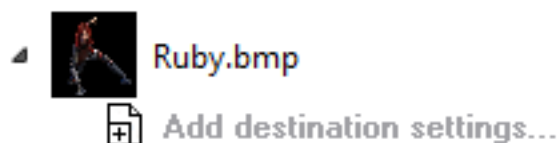
A new window will be displayed
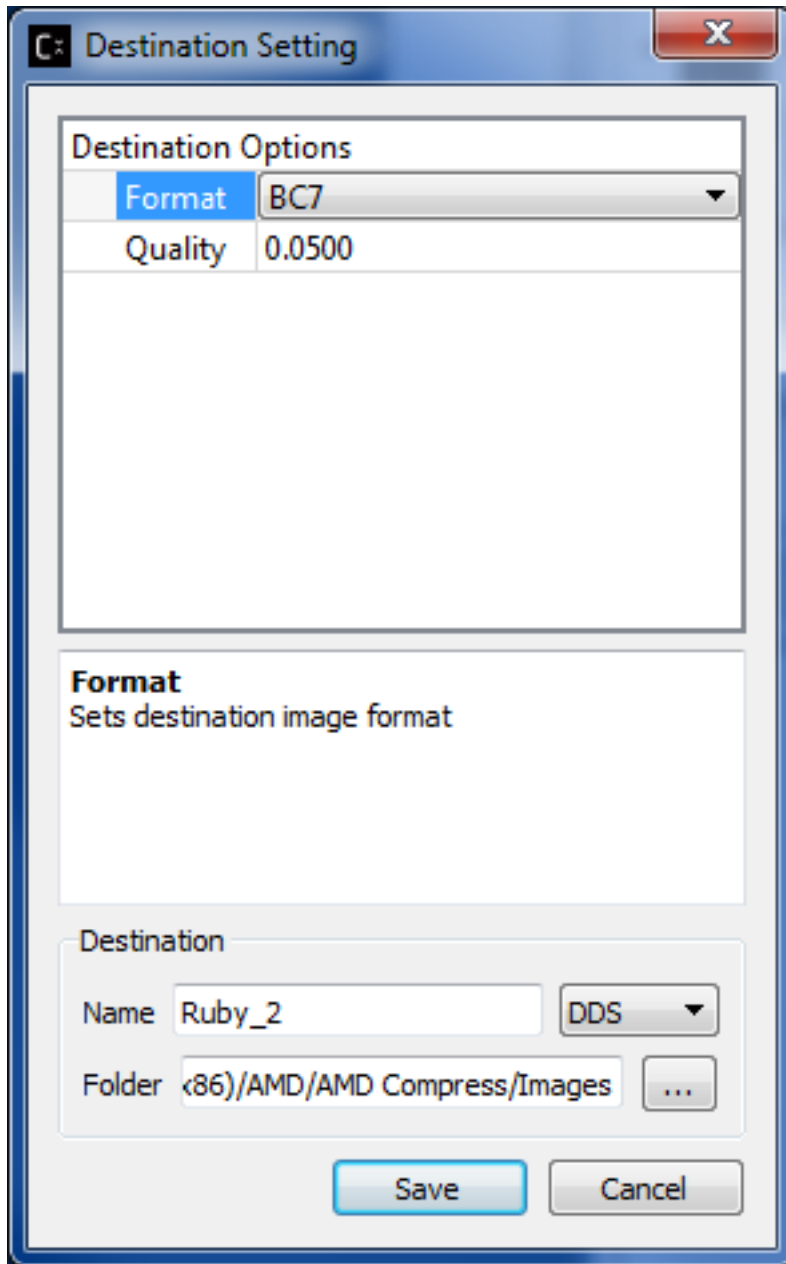
Add Destination Settings Window

Once you have set the desired options, the destination file name and folder; select save. This will now add the new item to the Project Explorer view.
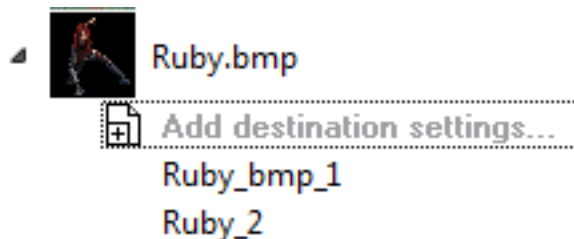
Note: In some cases, a red circle with a cross is displayed indicating that a file already exists and will be overwritten with the new settings. The current release does not check for duplications during setting.

Using Codec Quality Settings

BC1, BC2, and BC3 have discrete quality settings, These settings are available in the following ranges (varying the q setting in these ranges will have no new effects, q is a discrete coarse setting)

```
q = 0.0     to 0.01 sets lowest quality and fast compression
q = 0.101 to 0.6 sets mid-quality
q = 0.601 to 1,0 set the best quality and low performance
BC4 and BC5 have no quality settings, no changes in quality will occur if set.
BC6 & BC7 have full q ranges from 0 to 1.0
```

Setting Global Quality Settings

Users can override all individual destination compression settings, using a globally set value before processing

Currently, only the quality settings can be overwritten with a new global setting.

The process is as follows:

On the project explorer click on "Double Click Here to add files"
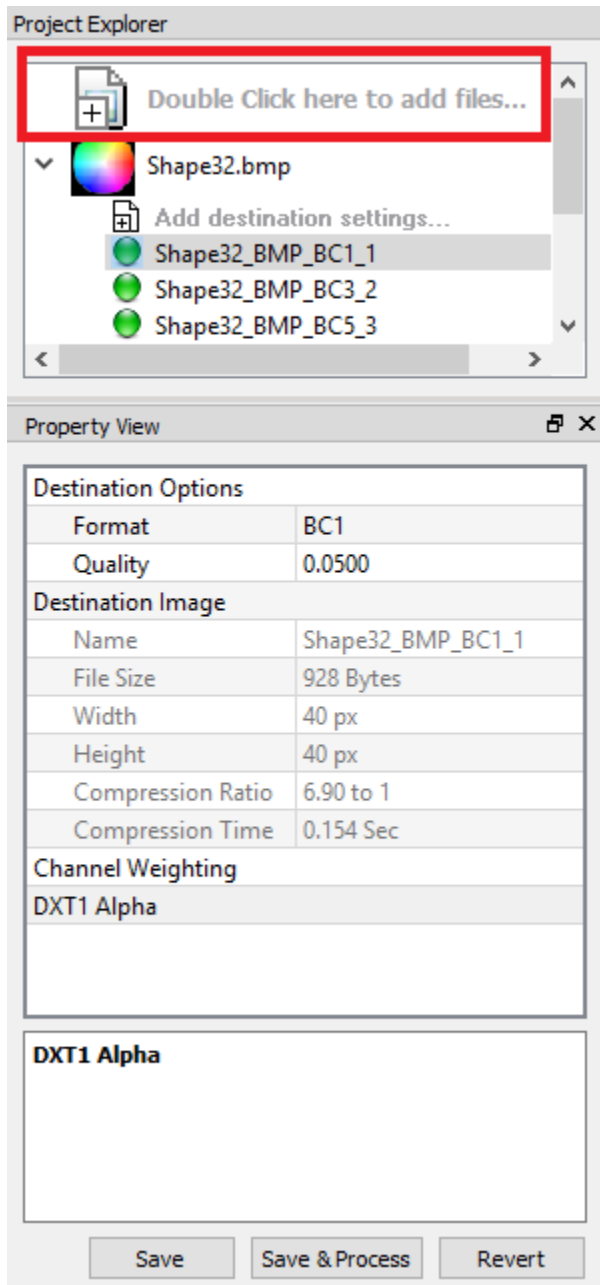
A new property view will be displayed
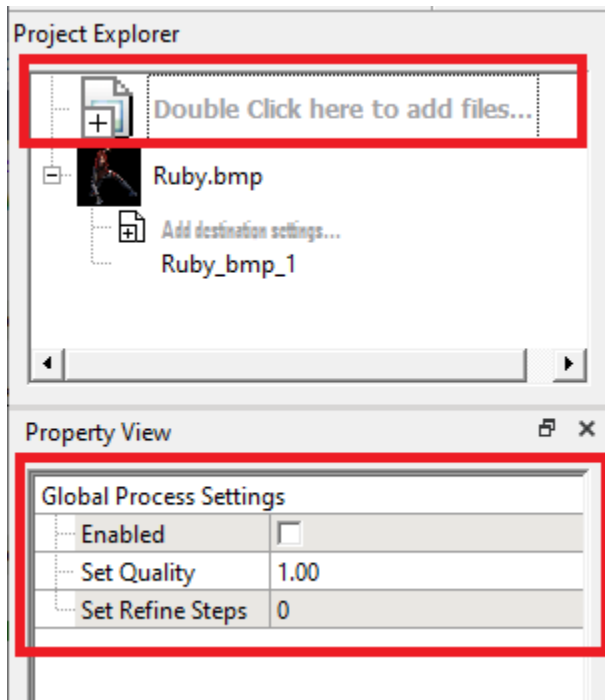
Set a new Quality value to override all existing quality settings for textures in the project explorer, a value of 0 with restore the old values and disable the global settings
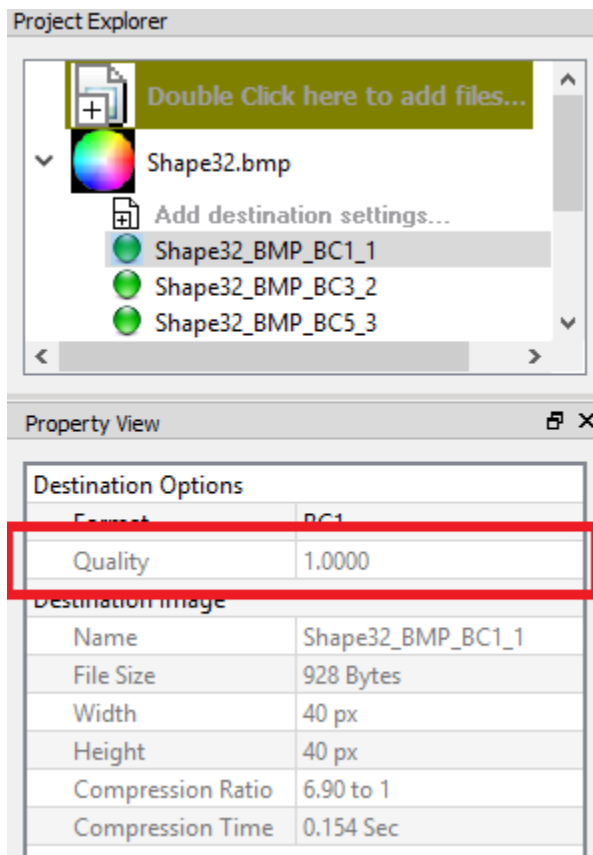
When an override is set the textures will display the new override setting and disable its editing features.

Notice also that the "Double Click Here to add files" background color has also changed to indicate that an override setting is in effect, it will return to a white background if the override settings are turned off.

Message Output

A message output window is displayed when an item on the Project Explorer is processed for either compression, format transcoding, or mipmap level generation.

It displays information on the process that was just performed and may contain details such as performance, results, or error(s) if a process was unsuccessful.
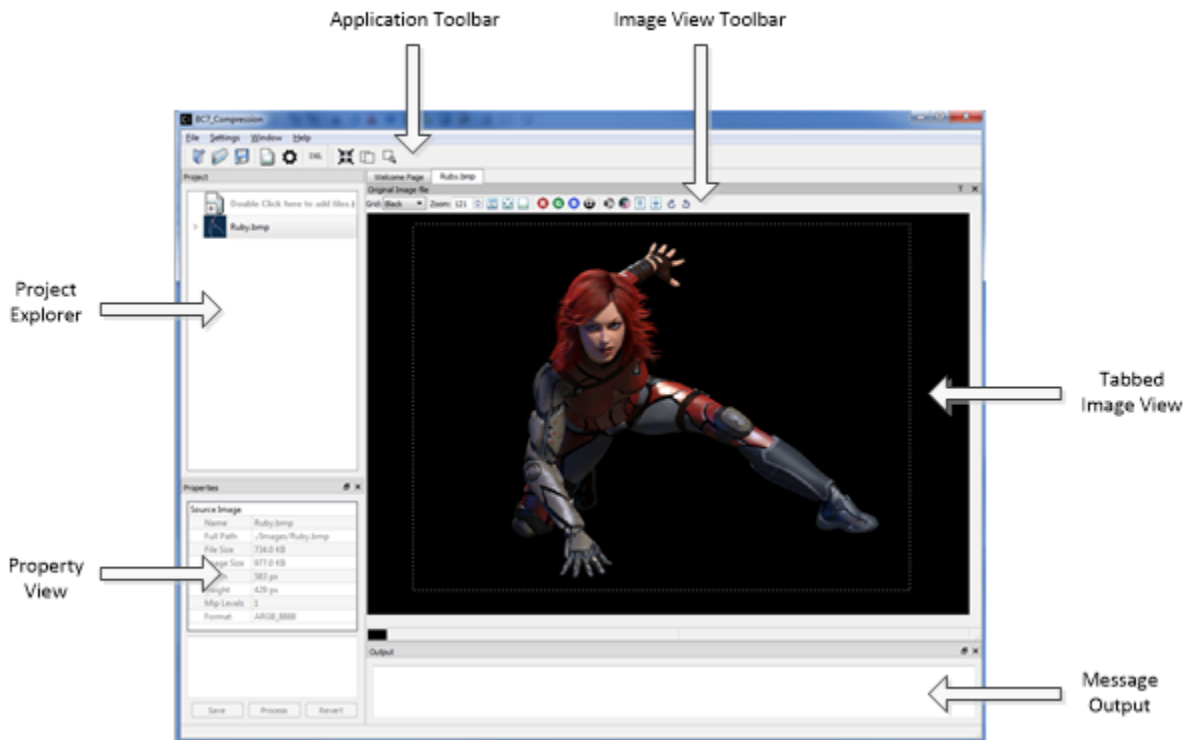
Detaching Windows from the Application

With the mouse position next to any gray shaded area next to the title of the window, press down the left mouse key and at the same time slide the mouse cursor away from the application. This will detach the window and allow you to position it at a new location, either inside the application or at a new desktop screen location. Once you have positioned the window to where it is desired, simply release the left mouse key.

Above are illustrations of where the mouse can be positioned for moving the window.

Note: The Project View window is not moveable.

When the window is moved "Un-Docked" around the application a blue shaded area will be displayed showing what new places the moved window can be placed.

Window with Blue shaded application section.

Example of Window outside of the application.

The advantage of moving windows outside of the application is to provide a larger viewing area for the image and allow side by side comparisons when using multiple monitors.

Batch Processing from Command Line Tool

Once a project has been setup, just like processing from the GUI you can also setup processing using the Compressonator CLI command line tool. The Command line tool has more options that are currently not supported on the GUI application. While 3D mesh process is not supported on Command line tool yet.

Once the Batch file are generated from the GUI, it can be edited to include more options used by the command line tool. This also facilitates automated generation of compressed files from many source textures.

Steps to generate a batch file:

- Creating a project file and set up the desired destination settings.

- Open the file menu and select "Export to batch file..."

- Select a destination file name and location to generate a batch file that can be run from command line.

- Select Save.

The batch file requires the CompressonatorCLI.exe, support file folders and DLL to be present on the same location.

The following files are required to run CompressonatorCLI.exe with the batch file:

... Include all files and subfolders under that folder

\plugins...	Qt Windows DLL and Qt Image Plugins
CompressonatorCLI.exe	Command line Application
qt.conf	Specifies the plugin folder
Qt5Gui.dll	Qt Run time DLL's
Qt5Core.dll	
libGLESv2.dll	
libEGL.dll	

'

3.2.3 Texture Compression

This page will go over the steps needed to compress an image using Compressonator GUI. It is assumed that you are already familiar with the basics of the application and how to use the sample projects. If not, please refer back to the *Getting Started with Compressonator GUI* page.

Compressing Images

After adding the *destination settings* for the image, select the preset setting and click on "Process" button or right click over the preset setting and click Process <name of the compressed file>
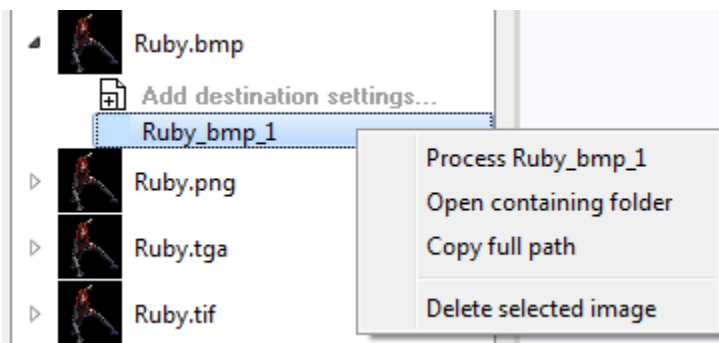
If you want to process all the settings that are set for an original image, right click over the root image and click Process all setting for <Original image file name>. For instance, example below will process and generate result for Ruby_bmp_1 and Ruby_2.
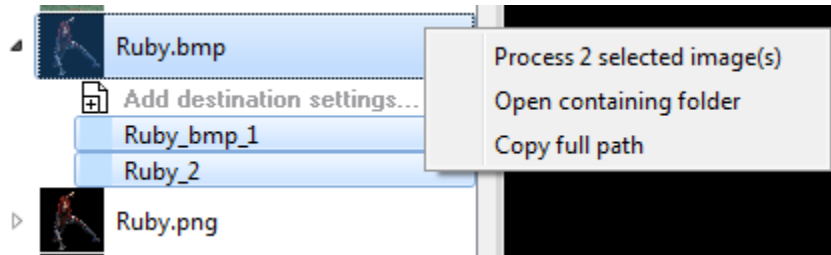
When compression process begins, a Progress window and an Output window will appear.
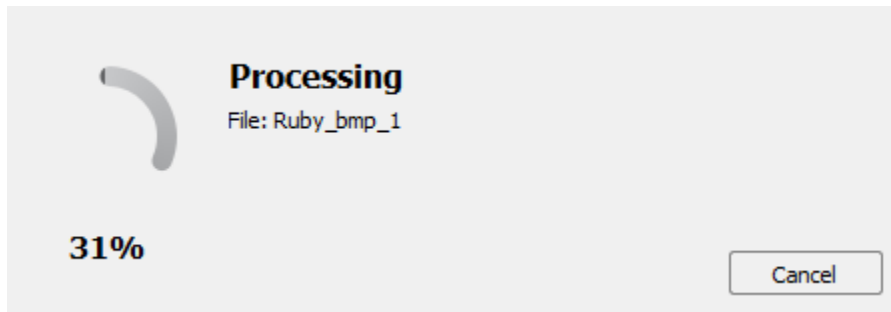
When the compression process completes, the Project Explorer will change to indicate the status of the resulting compressed Ruby_bmp_1 image with a small green (succeeded) or red circle (failed), and the Output window will indicate additional information on the succeeded or failed compression process.

Output

```
Processing: Ruby_bmp_1
CMD Line: ./Images/Ruby.bmp  ./Images/Ruby_bmp_1.DDS  -fd BC7  -Quality 0.0500
Source Texture size = 1000428 Bytes, width = 583 px  height = 429 px
Destination Texture size = 252288 Bytes   Resulting compression ratio = 3.97:1
Processing destination
Compressed to BC7 with 1 iteration(s) in 0.451 seconds
DeCompressed to ARGB_8888 with 1 iteration(s) in 0.141 seconds
Total time taken (includes file I/O): 0.599 seconds
Done
====== Compress: 1 succeeded, 0 failed ======
```
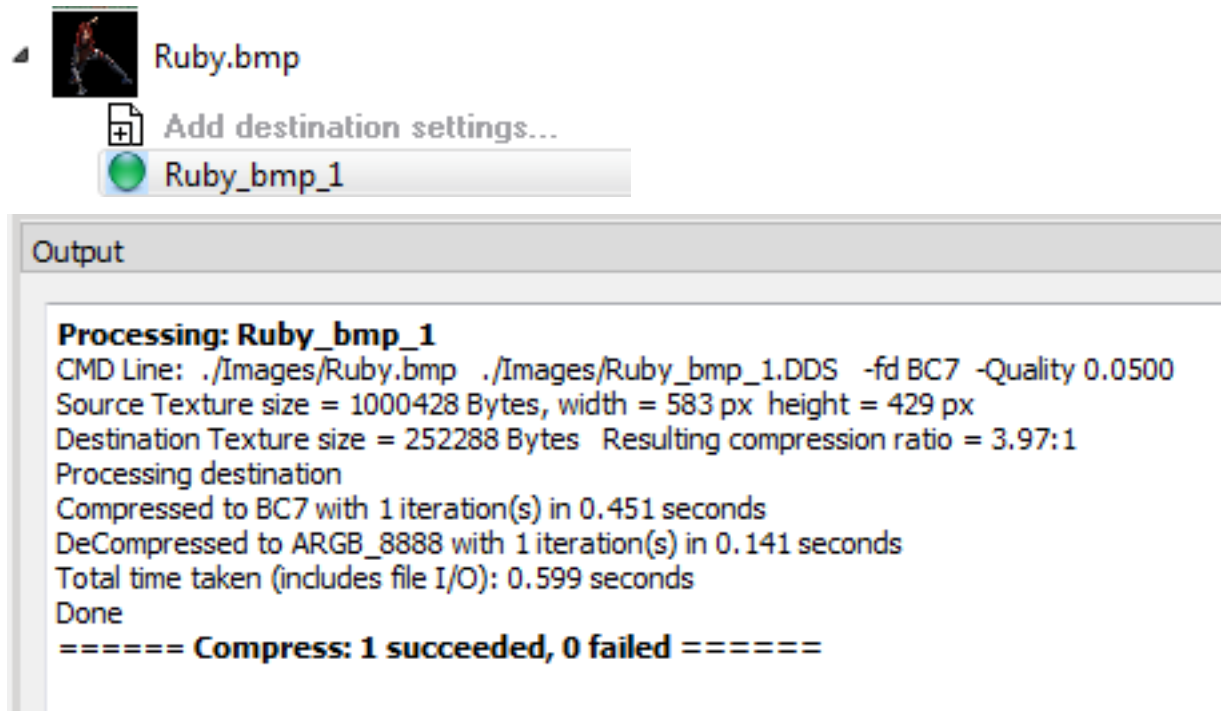
Converting Image Formats

Converting an image to a different format of similar or like quality (Transcode)

1. Add the image to the project explorer

2. Add a new destination item setting with a format of ARGB_8888 (applies to most images) unless the original image is HDR in which case select ARGB_32F.

   Note: The application cannot currently convert between source and destination variations of ARGB_8888 and ARGB_32F formats

3. Now set the desired destination file extension from the supported list

4. Process the destination item

Using the Make Compatible Feature

With the GPU or CLI tools, users can compress HDR and LDR images using any BCn codec without specific knowledge of the source format. For example, HDR images can only be compressed by BC6 any attempt to use BC1 to BC5 or BC7 will fail compression. Transformations are automatically handled to make the image source format compatible with the encoder when using the CPU or GPU based encoding.

Compressonator performs auto conversions of FP16 to Byte and Byte to FP16 formats when encoding textures with GPU or CPU encoders. A pre-conversion of the source data is performed into a temporary buffer which is then sent for processing, once the processing is completed the buffer is removed.

To see how this works, try processing an EXR file format image to compress with formats like BC1 to BC5 or BC7 using the "Encode with" option for HPC, GPU_DirectX or GPU_OpenCL If you are unfamiliar with how to process textures, check the tutorial on "getting started using sample projects".

Compressing Signed Channel Images

Signed channel component processing is available in the Compressonator SDK, Command-Line, and GUI applications.

This supports processing textures from 8-bit unsigned or signed channels to BC4 and BC5 formats.

In the GUI the process is no different than that described for compressing images using the project explorer. Image views will show additional status information about the signed channel data and how it is rendered onto the image views as unsigned channel formats.

For BC4 and BC5 encoding as a signed component use the destination settings option as BC4_S and BC5_S

Decompressing Textures

Compressed images can be added to the project view as original items and viewed with the Image Viewer.
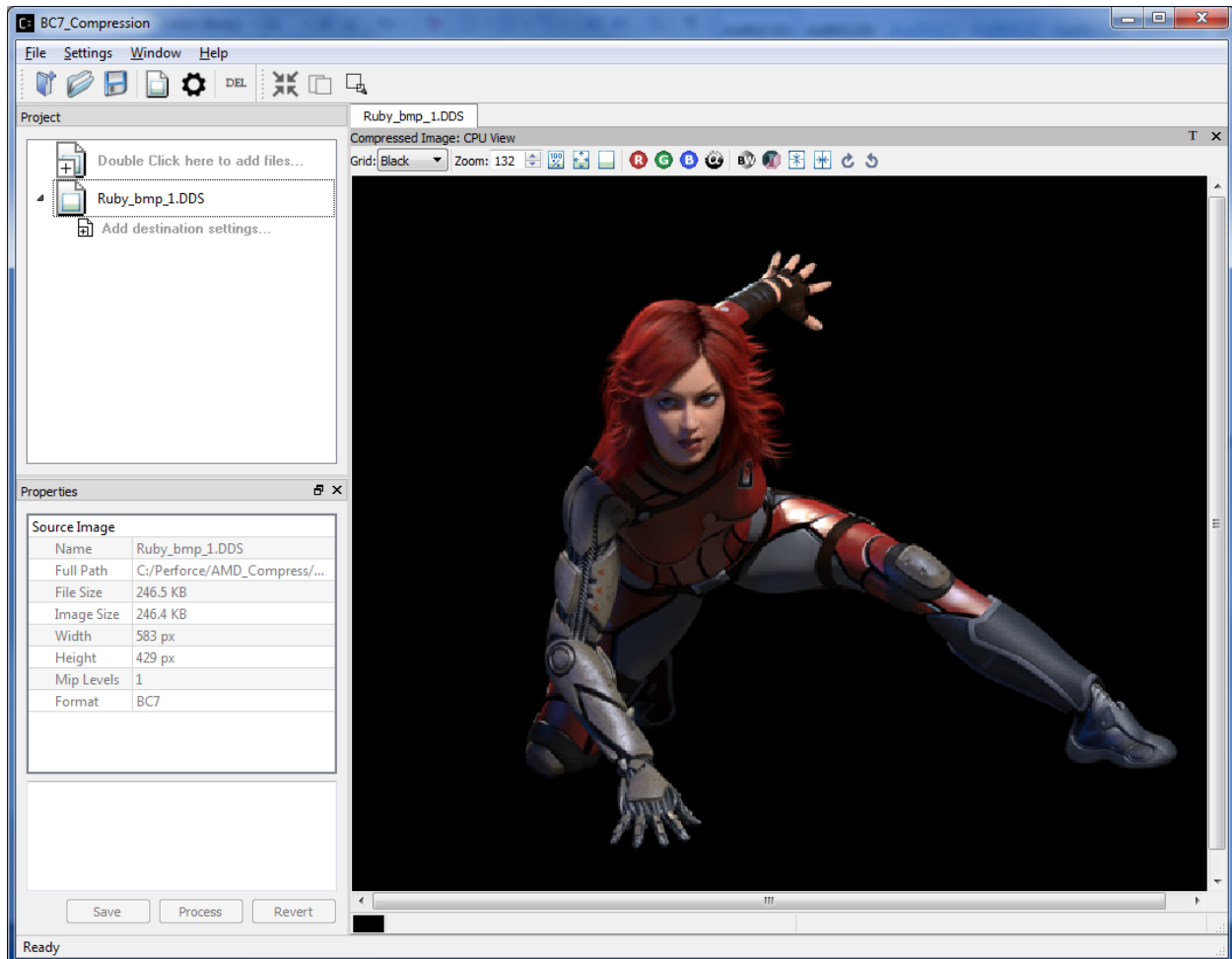
BC7 Compressed Image View of Ruby view as original image

Additional destination item settings can then be added to decompress the original item to a new format such as BMP, PNG etc.

Note: Compressed images used as original image cannot be recompressed to a new compressed destination, the selections are still enabled for support in future versions.

In the example below, a BC7 compressed image ruby is uncompressed to ARGB 8888 format and saved as a KTX file.
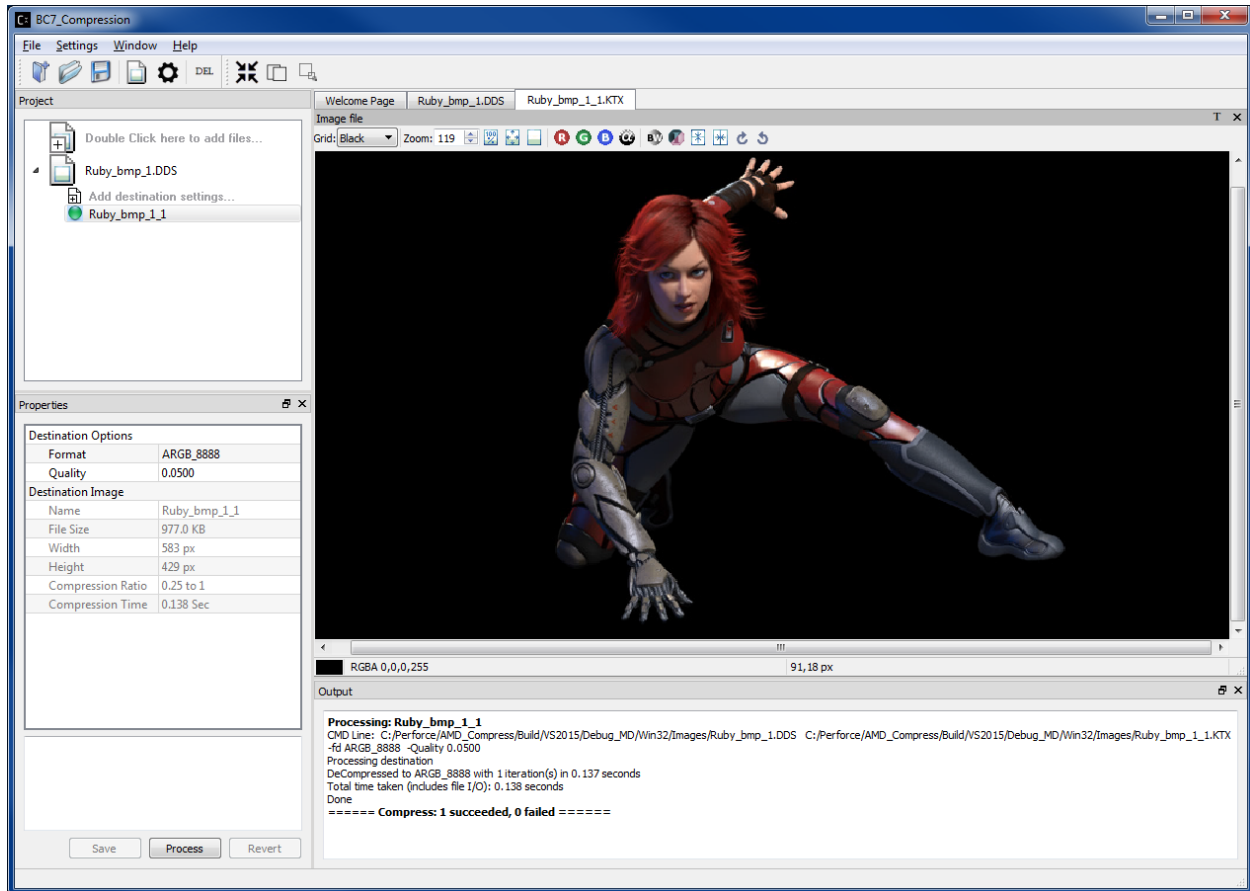
Ruby DDS file saved as a KTX file

Improving Compression Quality

The application allows multiple variation in processing a source image with a number of different image compression techniques. Each having specific quality and performance results.

By cycling different options between compress formats, quality setting and examining the image differences and views, users can quickly determine what works best for their image samples.

Channel weighting

Channel weighting option (X Red, Y Green and Z Blue weighting) enabled on compression destination setting for supported Compression Codecs.
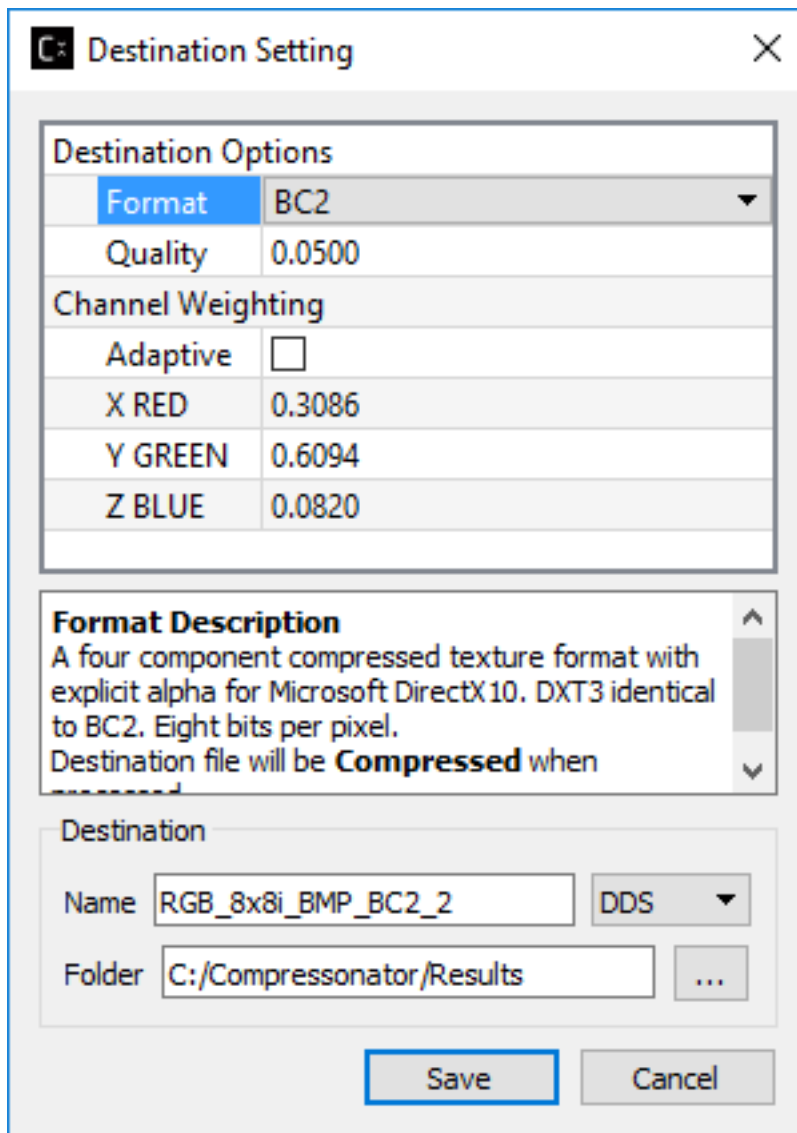
Users can edit the color channel weight ratio (total is 1.0000) by editing the value on each channel. Each channel has their default value (recommended value) set.

Alpha setting enabled for BC1 (DXT1)

Alpha setting enabled for DXT1/BC1 compression

For DXT1/BC1 compression format, users can choose to enable/disable the alpha channel in the compression. It is disabled by default.
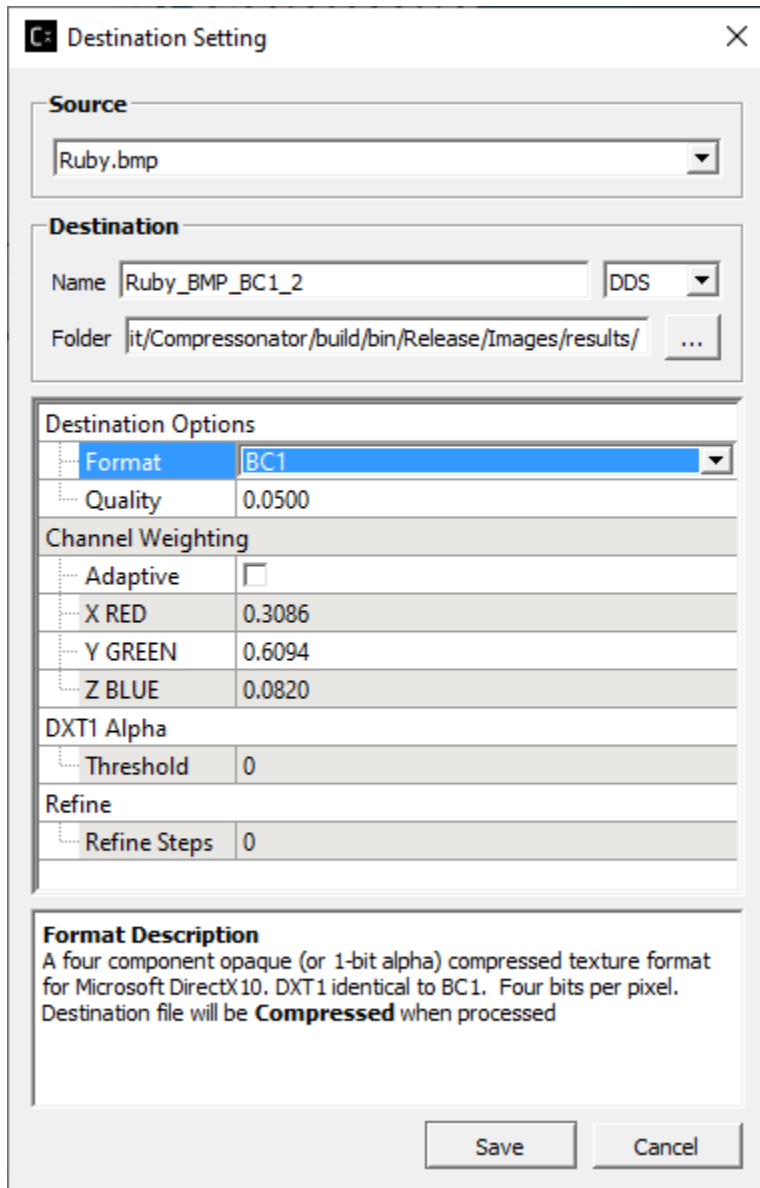
HDR Properties setting

HDR Properties setting enabled for half float and float type images (for HDR view support)

For HDR floating point images, users can choose to adjust the exposure, defog, and knee low as well as knee high properties of the display images. It is disabled by default.

It can be enabled and shown by choosing the "HDR Properties…" from the "View…" drop down list:
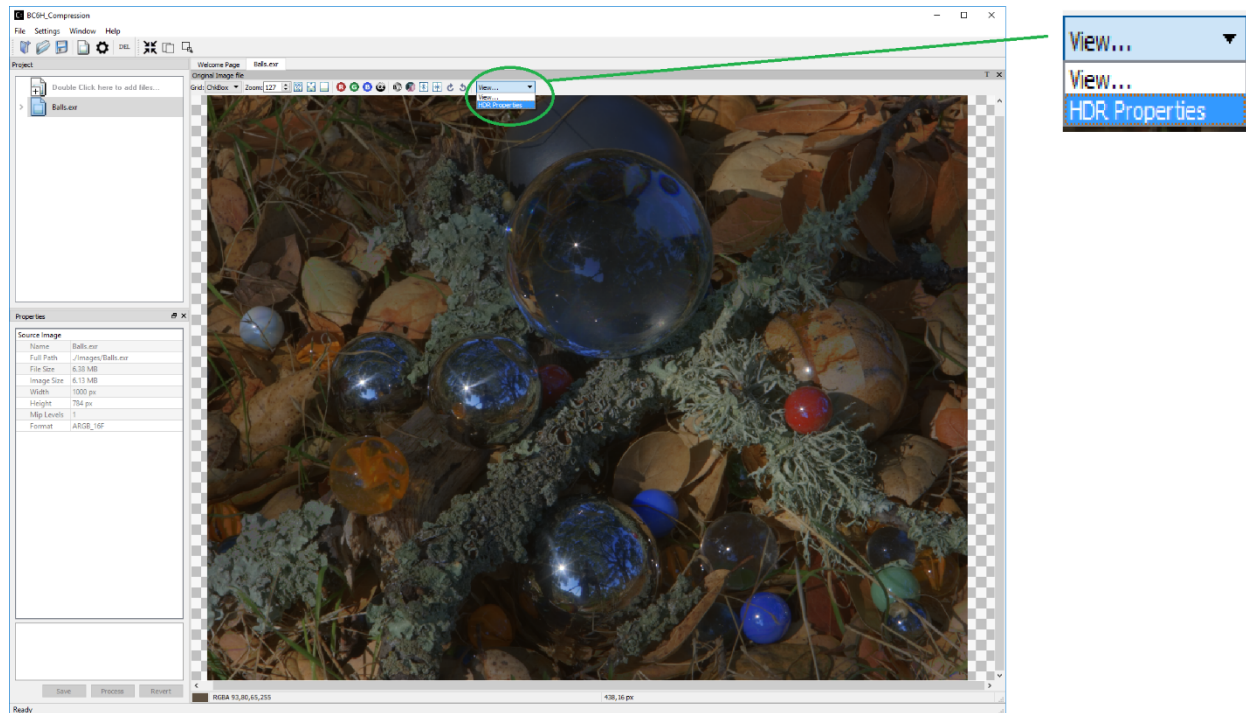
After clicking on "HDR Properties..." the window below will pop up:
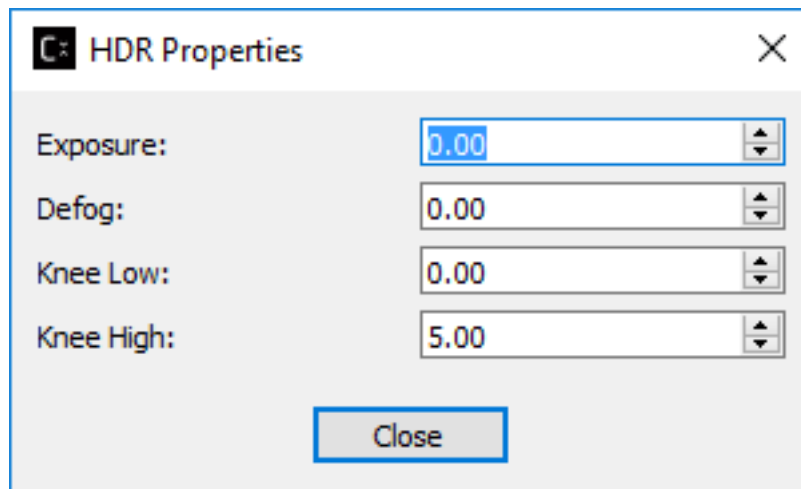
ETC2

Compressonator supports ETC2 formats that are based on the following standards:

CMP	Vulkan	DirectX	OpenGL
CMP_FORMAT_ETC_RGB	VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK	DXGI_FORMAT_UNKNOWN	COMPRESSED_RGB8_ETC2
CMP_FORMAT_ETC2_RGB	VK_FORMAT_ETC2_R8G8B8_UNORM_BLOCK	DXGI_FORMAT_UNKNOWN	COMPRESSED_RGB8_ETC2
CMP_FORMAT_ETC2_RGBA	VK_FORMAT_ETC2_R8G8B8A8_UNORM_BLOCK	DXGI_FORMAT_UNKNOWN	COMPRESSED_RGBA8_ETC2_EAC
CMP_FORMAT_ETC2_RGBA1	VK_FORMAT_ETC2_R8G8B8A1_UNORM_BLOCK	DXGI_FORMAT_UNKNOWN	COMPRESSED_RGB8_PUNCHTHROUGH_ALPH

The GUI and Command Line formats are specified as:

GUI and CLI	SDK
ETC_RGB	CMP_FORMAT_ETC_RGB
ETC2_RGB	CMP_FORMAT_ETC2_RGB
ETC2_RGBA	CMP_FORMAT_ETC2_RGBA
ETC2_RGBA1	CMP_FORMAT_ETC2_RGBA1

Process results for image with alpha channel:

Using the image sample ruby_alpha.tga, add compression setting for ETC2_RGB, ETC2_RGBA and ETC2_RGBA1 and process.
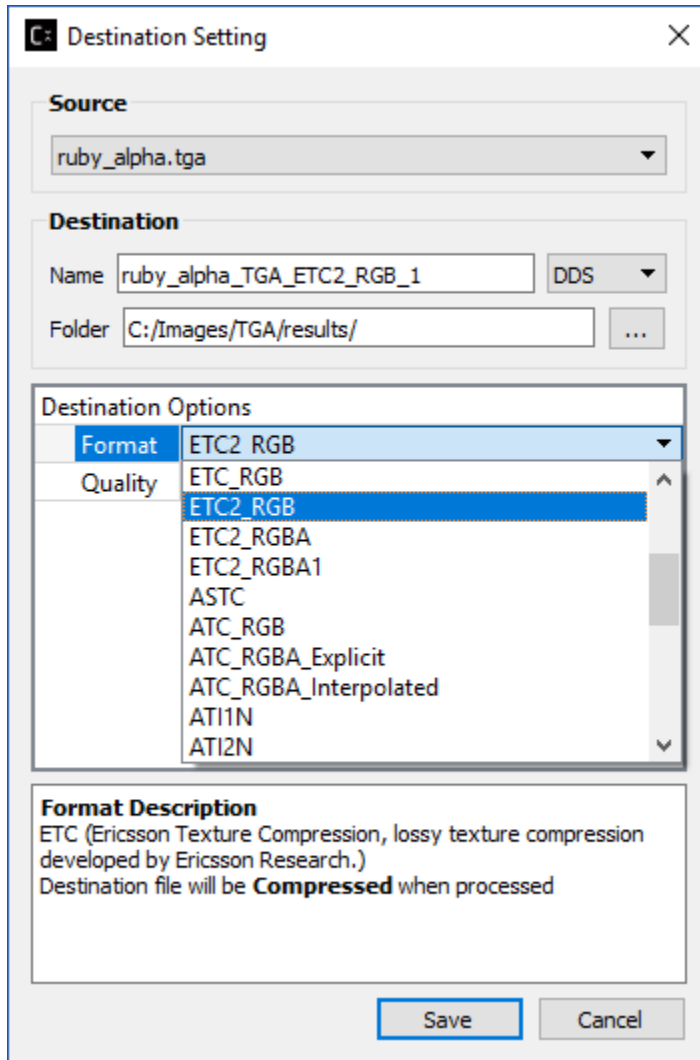
The results should look like the following when viewing the decompressed images:

Ruby Image contains alpha channels and processed with ETC2_RGB

Ruby image processed with ETC2_RGBA

Ruby image processed with ETC2_RGBA1

Selectable GPU or CPU based compressed image views

You can select how compressed images are viewed on the GUI View image tabs. The compressed textures are decompressed into RGBA_8888 format using either the Compressonator CPU based decompression algorithms or the GPU via a common interface for OpenGL (version 3.3 and up) or DirectX (only version 11 is supported).

Click on ⚙ Set Application Options, Application Settings window pops up as shown below:

Users can choose to view the decompressed images using CPU, OpenGL, DirectX or Vulkan.
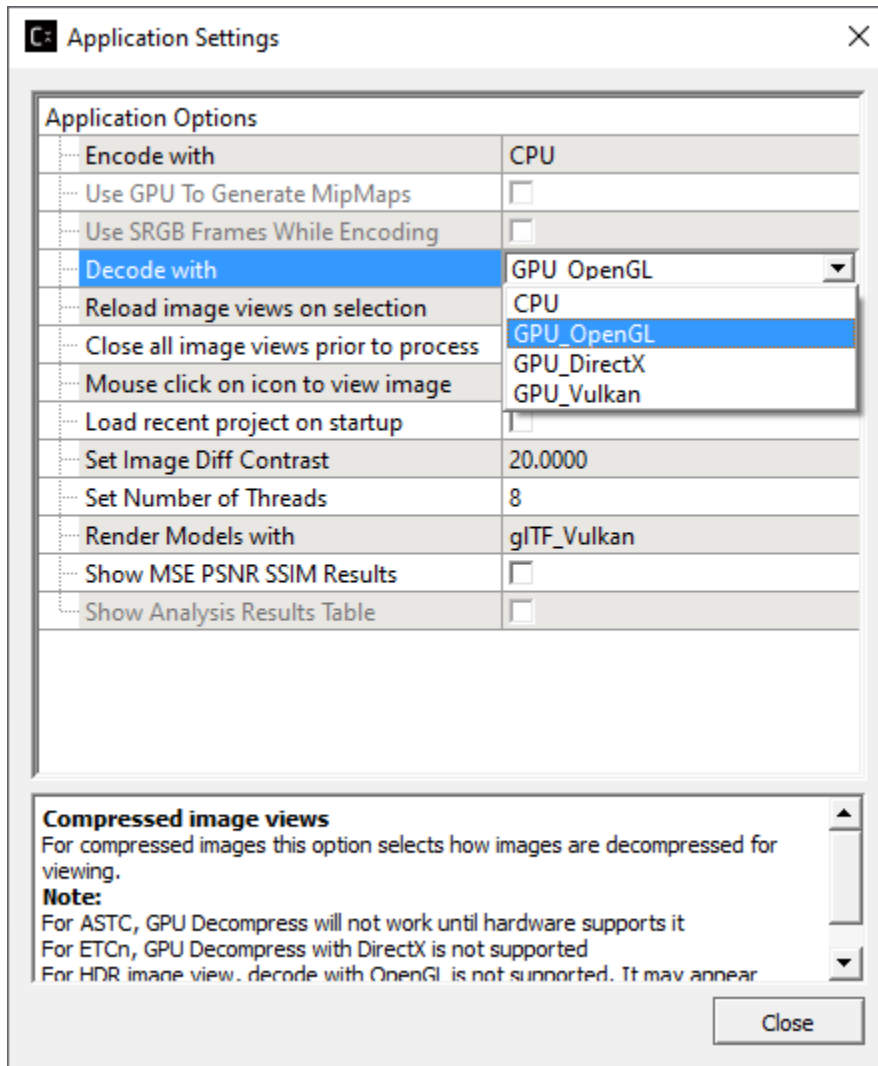
After setting the application options, click on the compressed image to view the image.

Example views of compressed BC7 image ruby_1.dds

CPU View
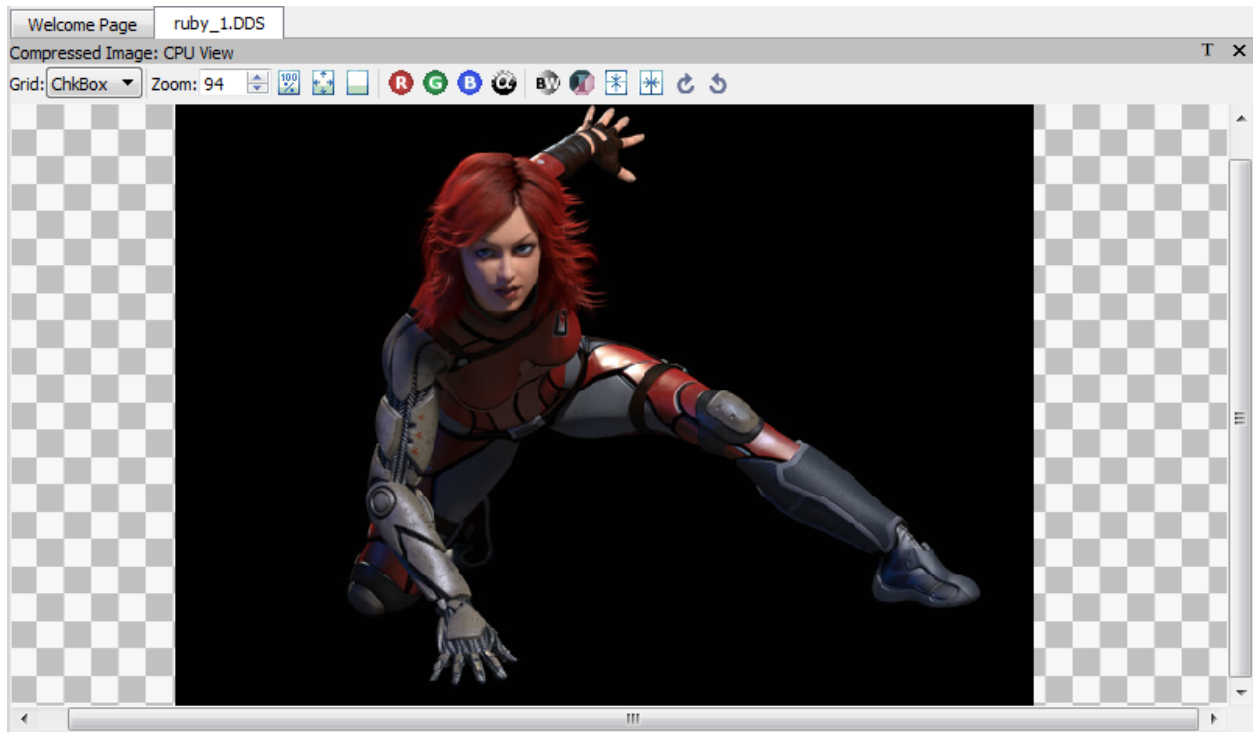
View using OpenGL
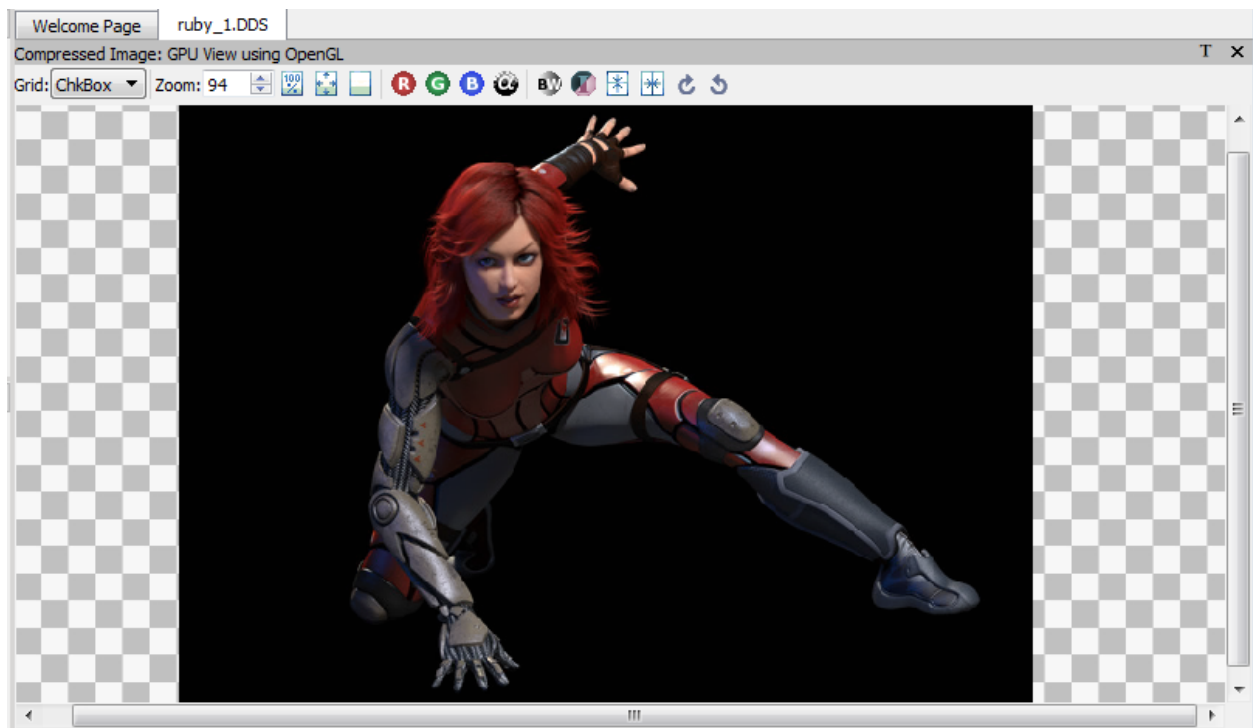
View using DirectX
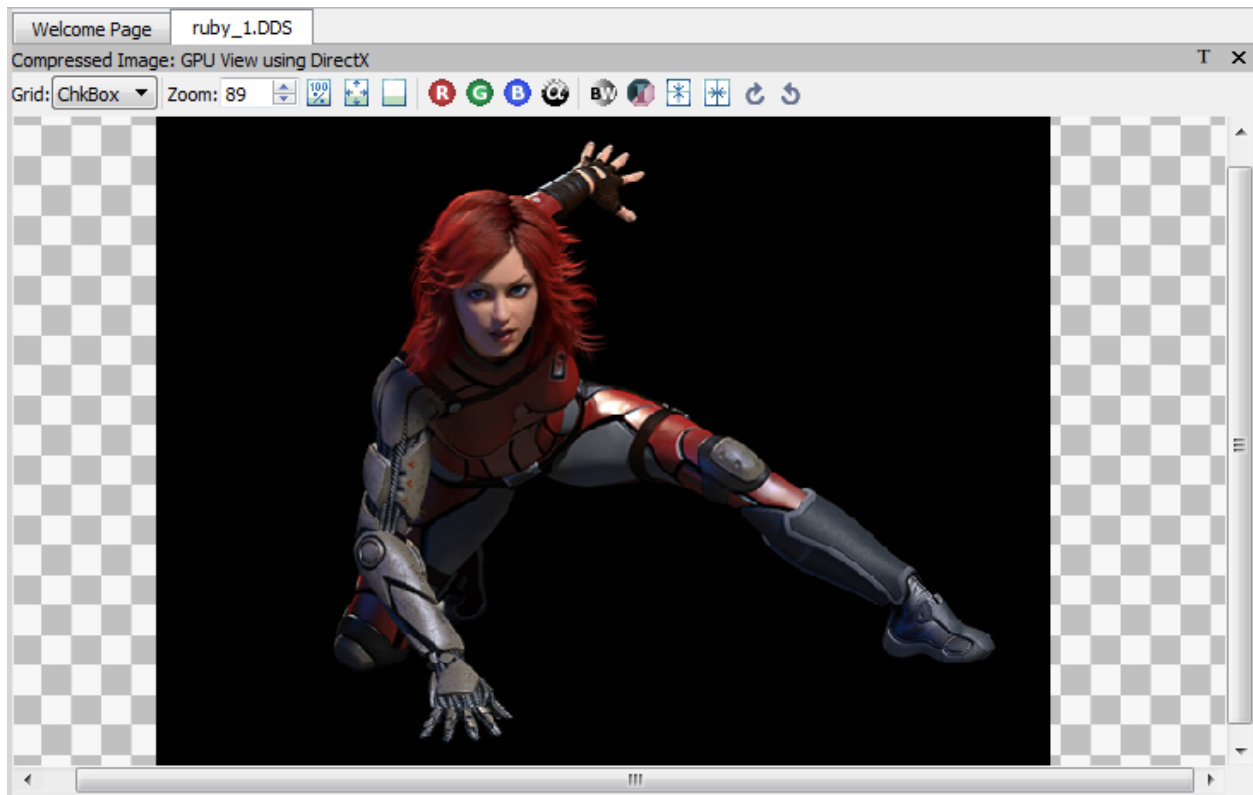
Users still able to generate image diff view with GPU as shown below:
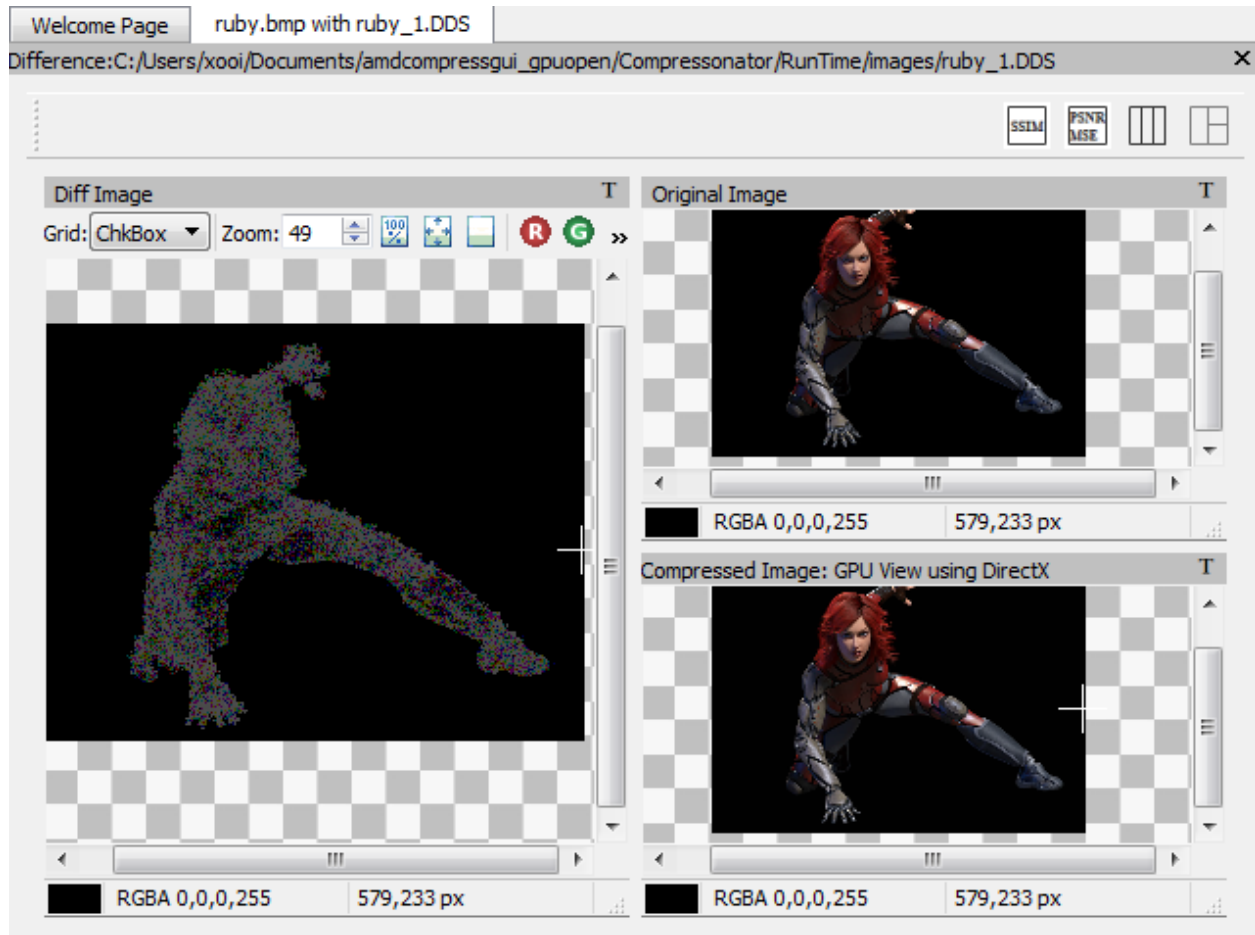
3.2.4 3D Model Compression

glTF Viewer and asset compression

Compressonator GUI application now supports glTF V2.0 file formats for viewing and processing. (Note: The CLI application does not support processing these files and will be updated in future revisions)

In order to use this feature the current V2.7 release requires **DX12 compatable HW and Drivers as well as Win10 RS2 or later**.

You can check if your system fulfills this requirement by running DirectX Diagnostic Tool (or dxdiag.exe) and refer to Win10 OS build 15063.xxx or later / Win 10 Version 1703 or later. Future releases will add support for OpenGL.

You can check by running windows dxdiag.exe on your system and check that it matches what is highlighted below.
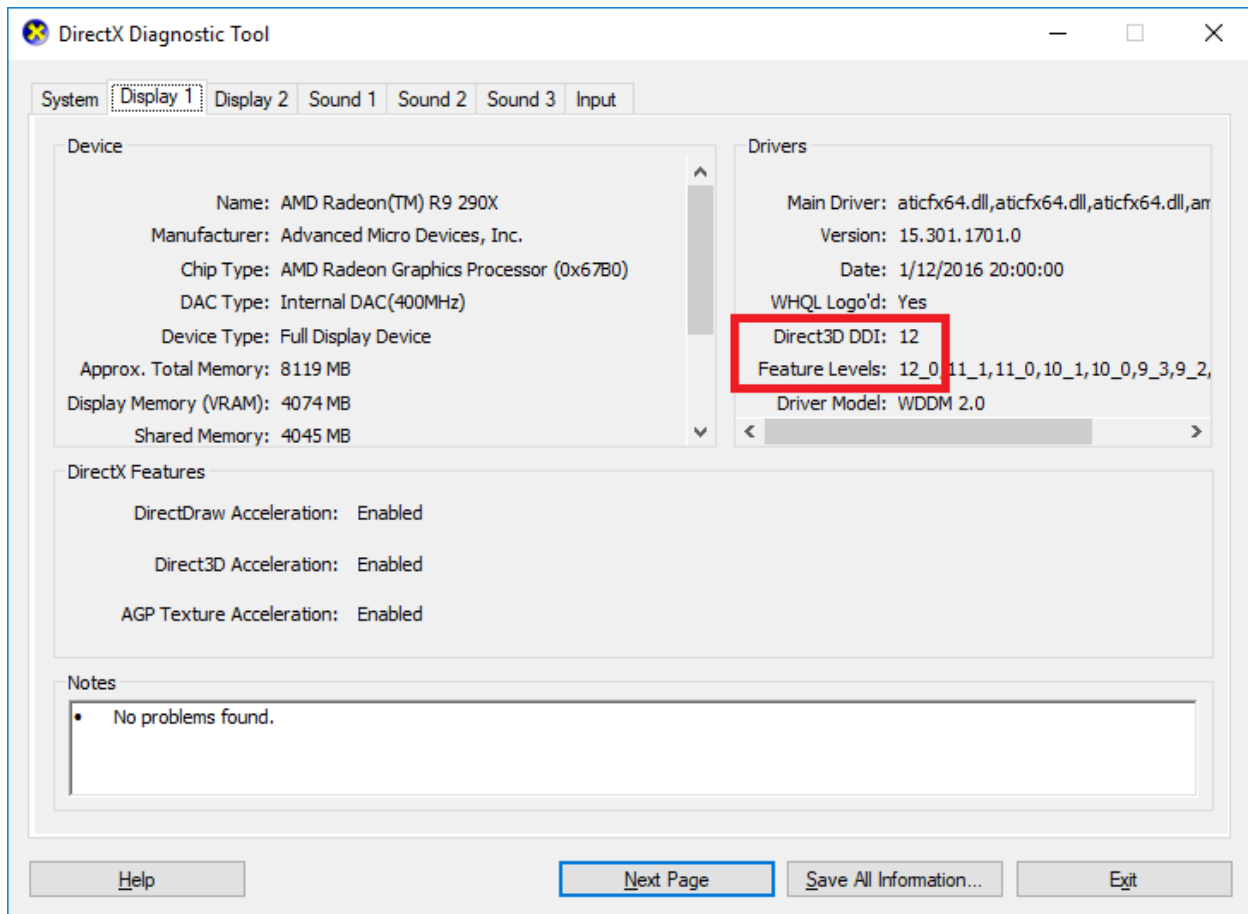
To use this feature,

Step 1: simply open a model with .gltf extension or drag and drop the file from a file explorer to the GUI Projects Tree view panel as shown below. The following steps is using the sample glTF file (SciFiHelmet.gltf) downloaded from https://github.com/KhronosGroup/glTF-Sample-Models/tree/master/2.0/SciFiHelmet
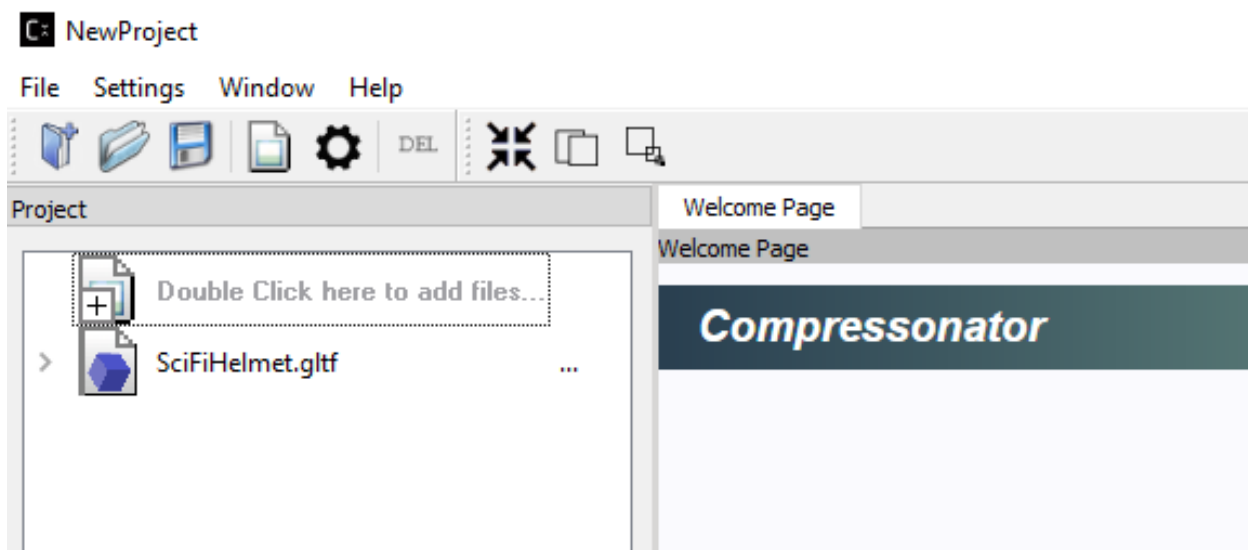

Now that we have a .gltf source we can now add any number of new .gltf files to use for processing.

Step 2: Clicking on (>) shows additional setting
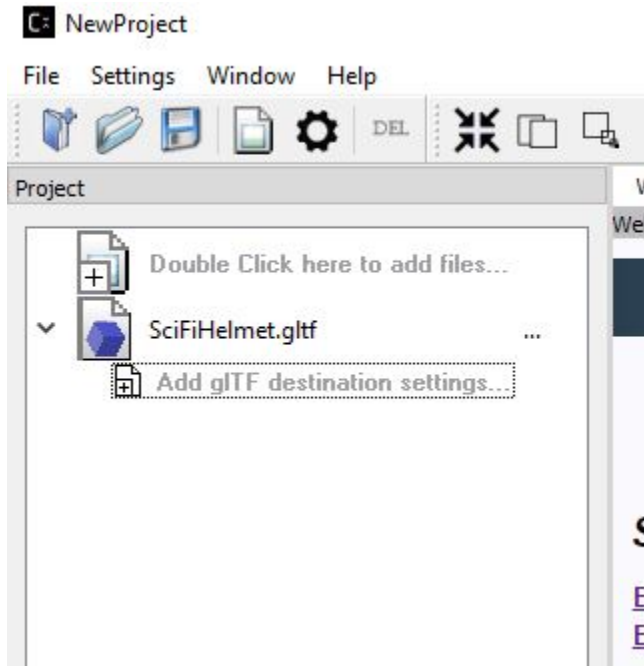
Double click on "Add gtTF destination settings" to allow copying the original file to a new file for further processing. A Destination Setting window will pop up as shown as below, this set up is used to preserve the original file settings and allow users to change only copies of it:
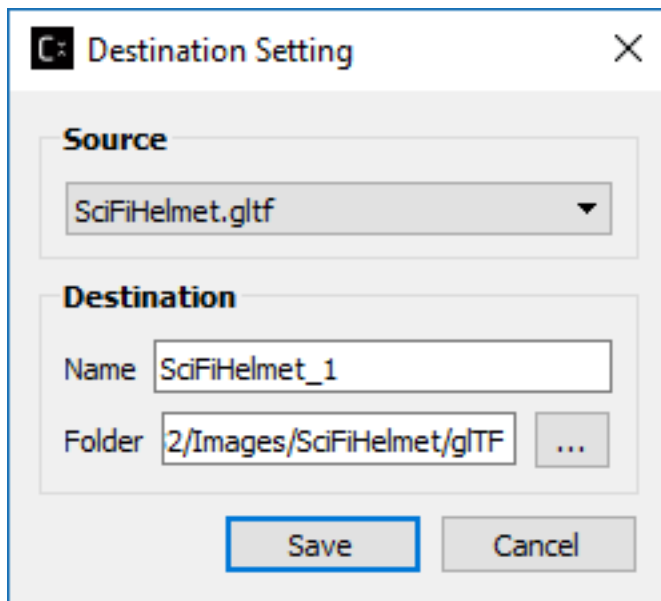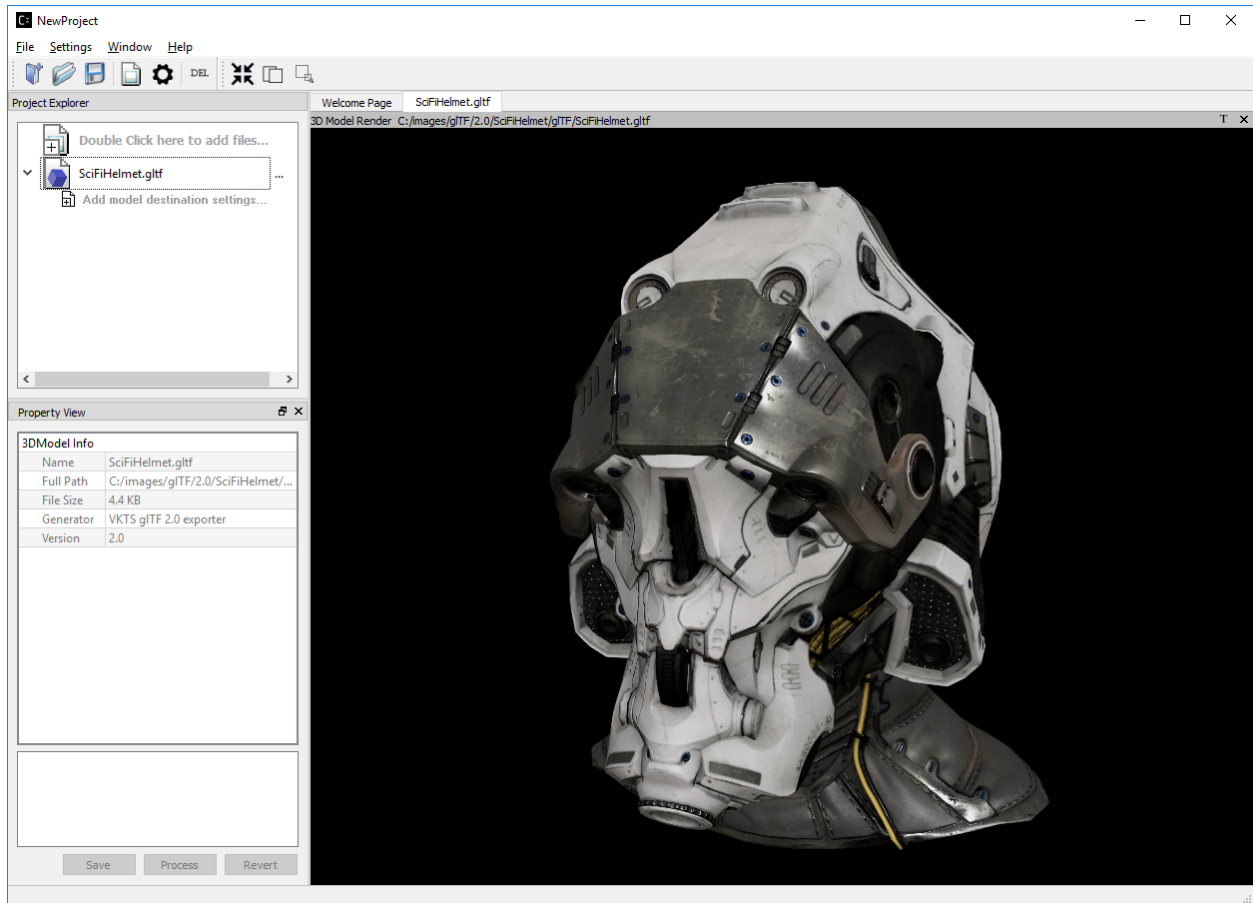

On this new settings dialog, you can change the default destination file name and folder then save

Step 3: The project tree will now display the copied gltf file. This copy still refers to the original sources images, cloud point data, meshes etc. . .
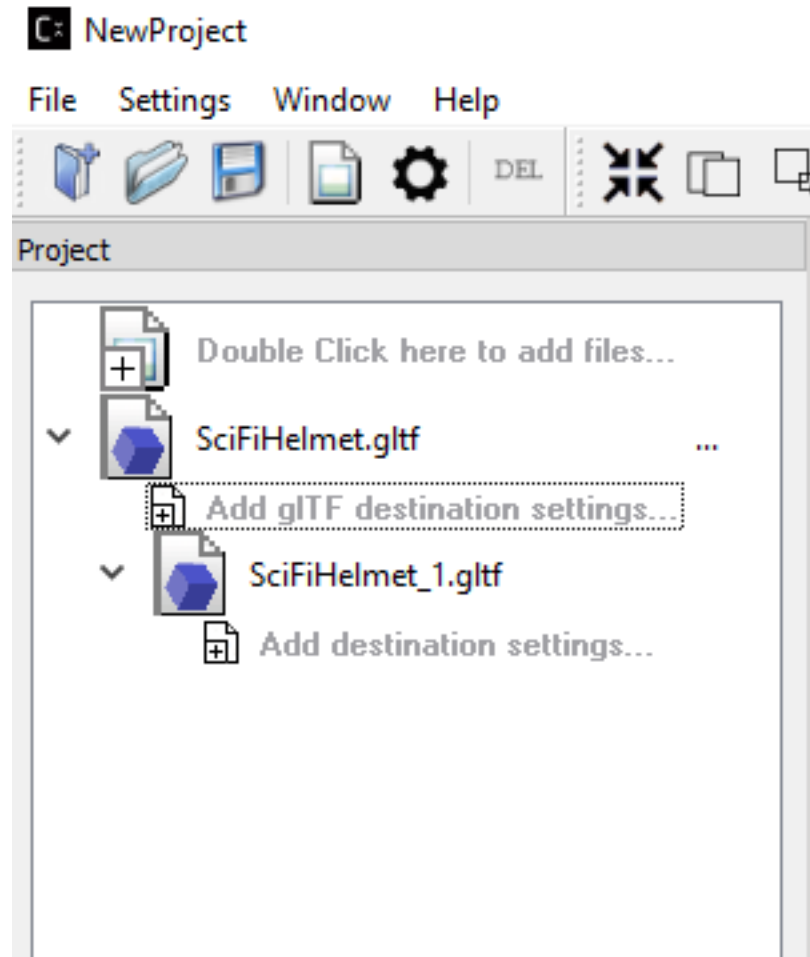
To view the 3D model simply click on the item or its icon, the 3D model view will show up in the viewer panel as shown below:
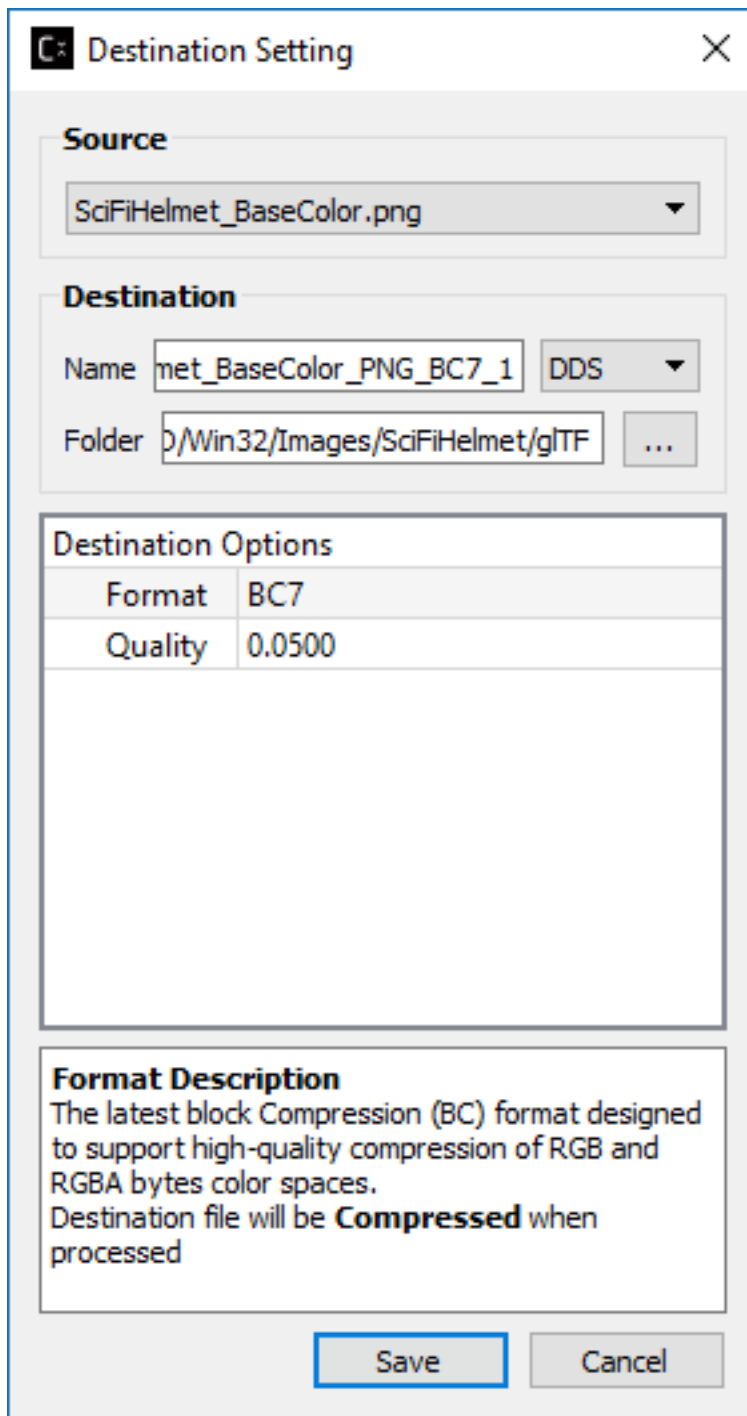
Note that the properties view has also been updated as shown above.

The 3D model view can be rotated and tilted by left mouse click and move on the view. To scale, use the mouse wheel in or out.

Note: By design all 3D views from any source are synced with mouse moves and scale. To reset the views back to origin, click on the right mouse button.

Step 4: Now that we have a copy of the original we are ready to change the source assets to use compression.

Clicking on (>) of the newly added glTF file to show additional setting. Double click on "Add Destination Settings." To show the Destination Settings Dialog as shown below:
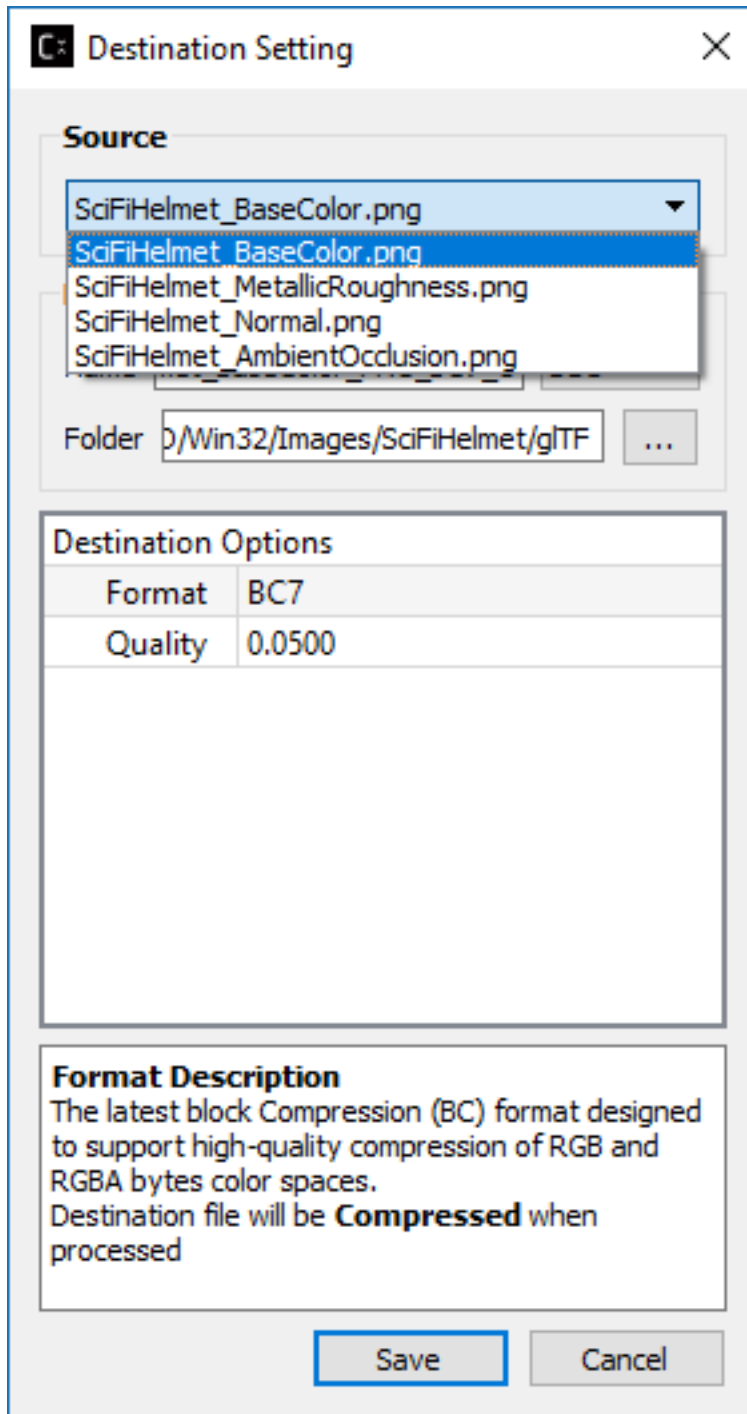
The Source drop down list will show all the files that are available for compression in the selected model.

Select the source file, then set its destination and compression settings.

Select save to mark the content of the gltf file "SciFiHelmet_1.gltf" to reference the new destination file.
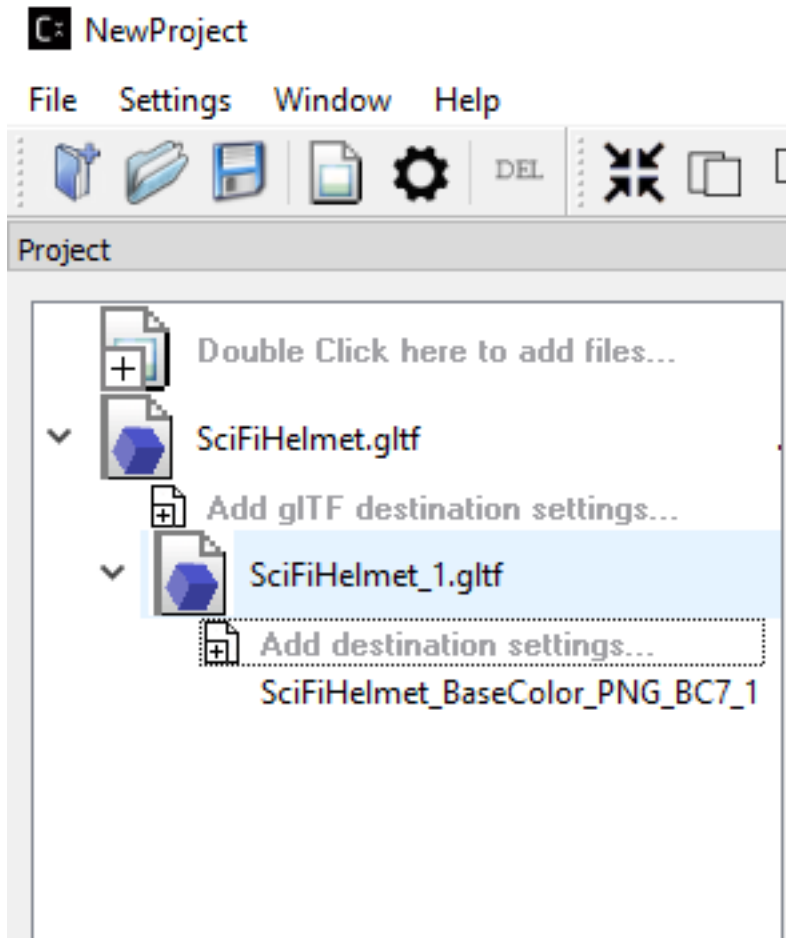
(Note since we have not yet processed the new compression settings the contents of the copy have not been updated. It will only update when the settings have been processed without any errors.)

Step 5: You can repeat "Step 4" process of "Add Destination Settings for all or any number of the remaining texture files that are not previously selected. Note that the Source drop down list in the Destination Setting dialog will get smaller each time a file is selected for processing as we only allow one format to be selected for each texture within the glTF file. If you would like to try out different format on the specific texture, please repeat Step 2 to 4.

Step 6: Right click on the tree item to process the new settings
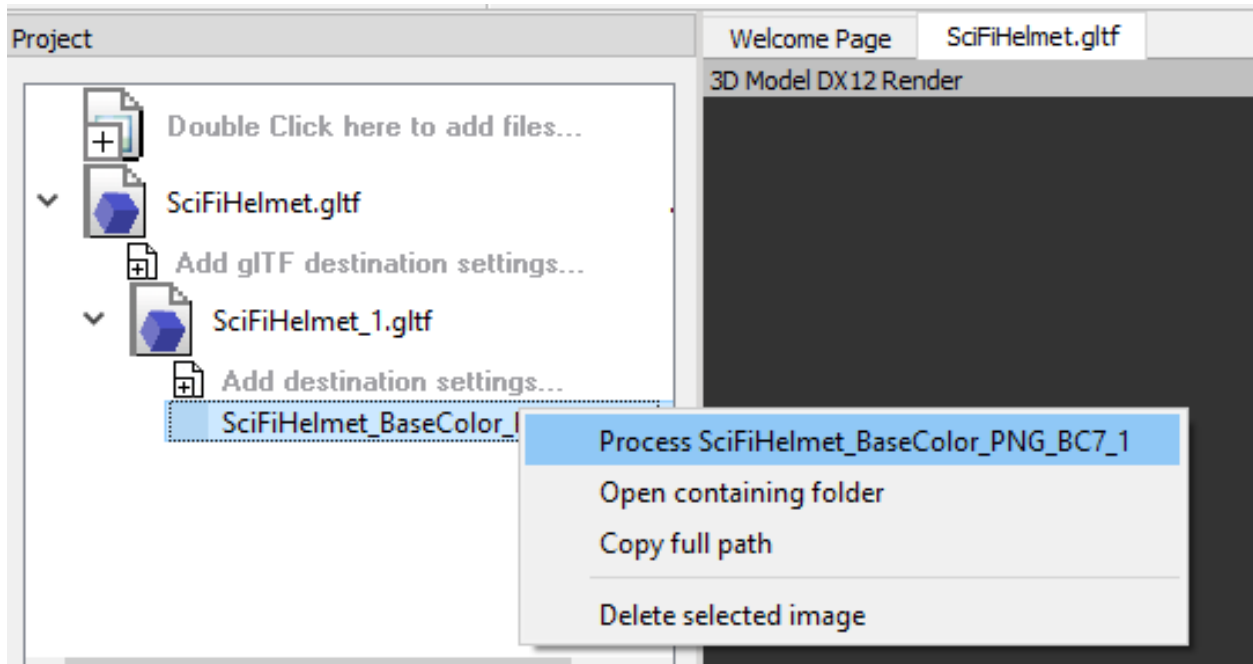
or select the process all icon on the tool bar.

Step 7: Once the process is complete and successful a new green icon will be displayed. At this point the SciFiHelmet_1.gltf file reference to the new destination file SciFiHelmet_BaseColor_PNG_BC7_1.dds file will be updated.
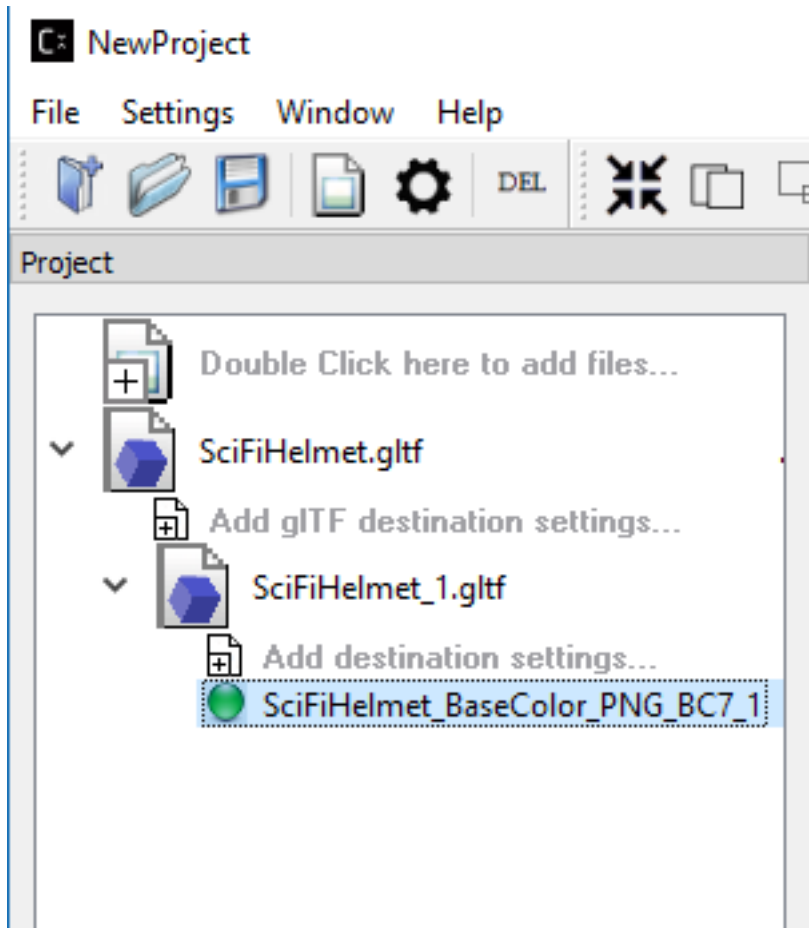
Vulkan® Viewer [Alpha Code] (Windows OS Only)

Allows viewing glTF files with textures. Currently features for "3d Model Diff" and "Stats" window are disabled.
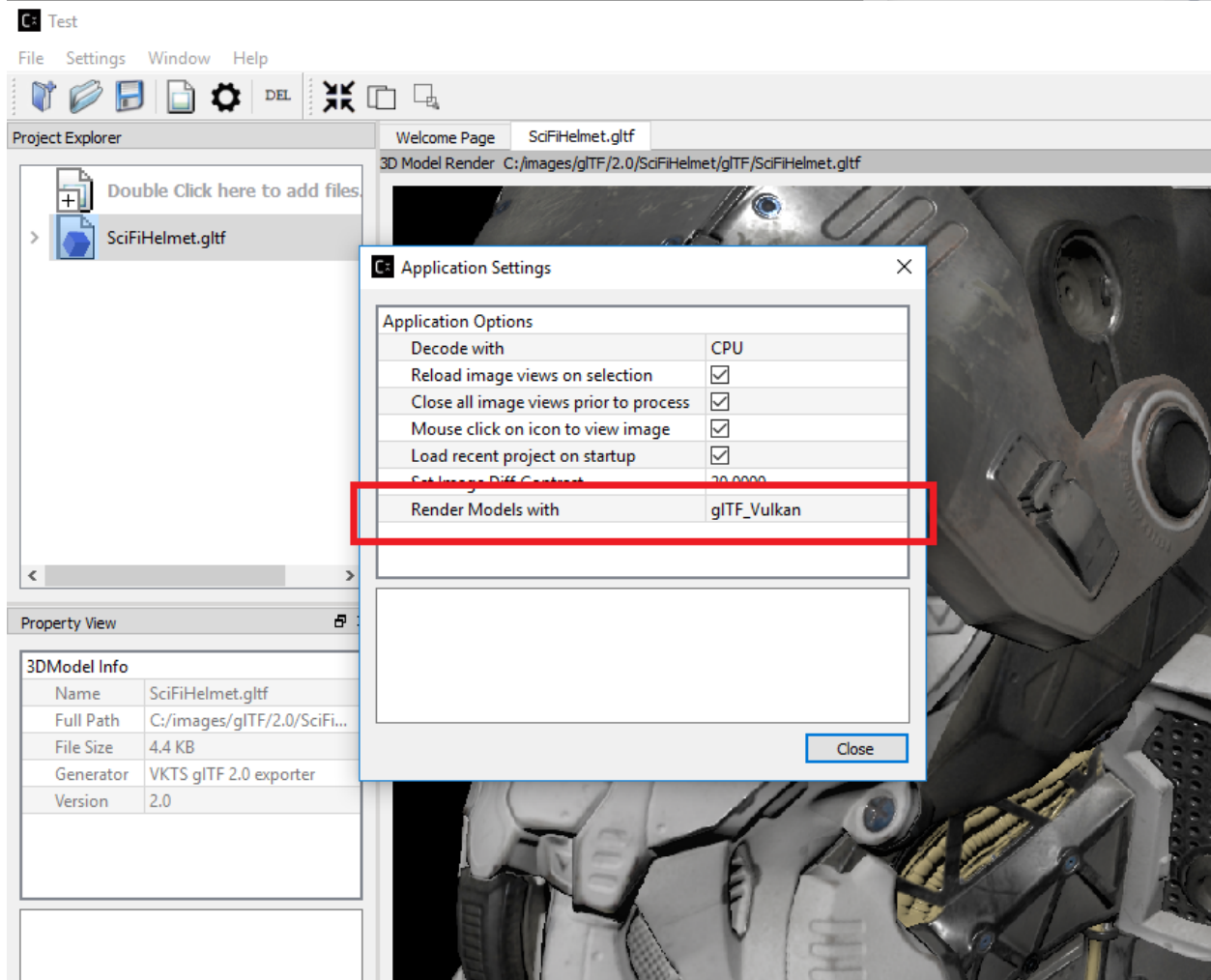
glTF 3D Model Diff

To view a 3D model diff, select the model copy "SciFiHelmet_1.gltf" and right mouse click on it. A new menu will display showing "View 3DModel Diff"
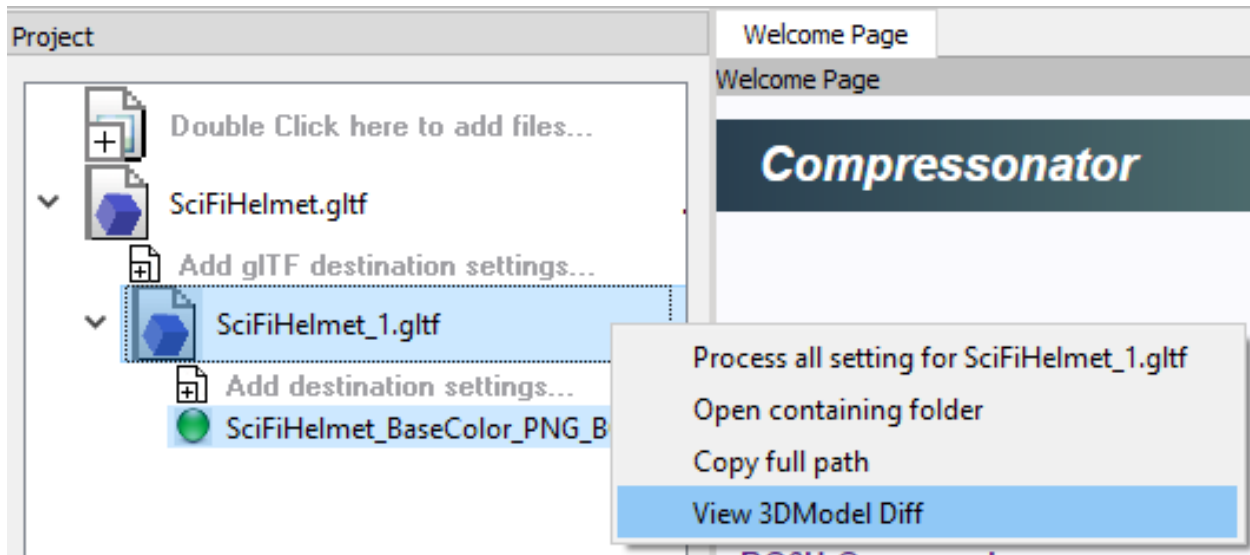
Once selected and after a few seconds of processing time. You will see the 3D Model image diff rendering of the original 3D model "SciFiHelmet.gltf" and it's Compressed Version "SciFiHelmet_1.gltf" alongside an animated render of both on the same view panel.

DirectX® 3D Model Diff View Modes

This feature allows the user to select a manual override of the automated difference view of two rendered models. The Auto View switches render frames at a predetermined rate after two render cycles of each model view, the Manual View allows the users to manually switch view using the keyboards space bar.
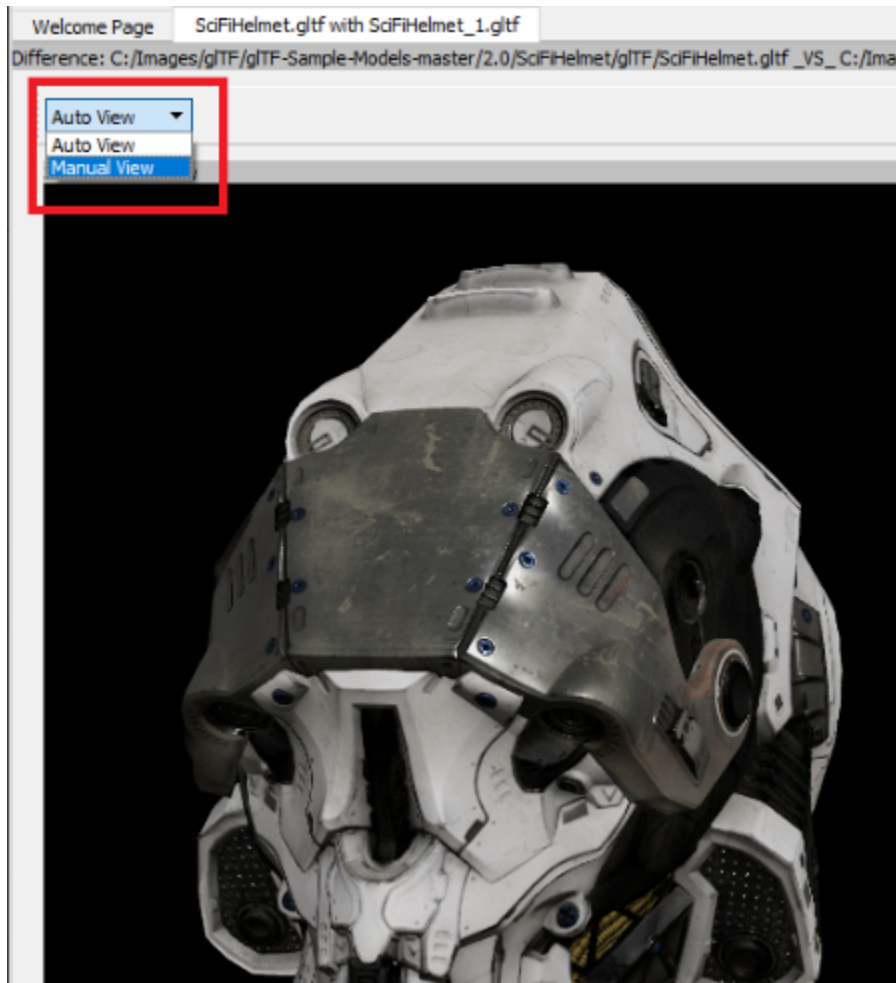
3D Mesh Optimization and/or Mesh Compression

Add 3D model file into Project Explorer by "Double click here to add files…" (recommended) or drag and drop into the Project Explorer. After that, click on the right arrow next to the model file added to expand the clickable "Add model destination settings…" view. Click on "Add model destination settings…" and click Save to add a resulted model file node as 2nd level of the Project Explorer tree view.
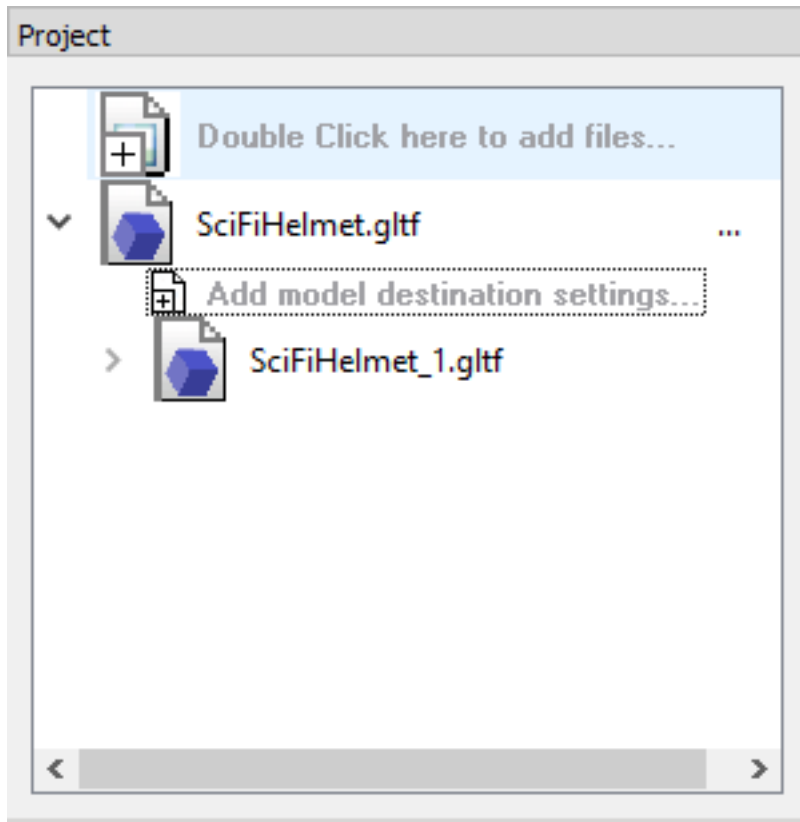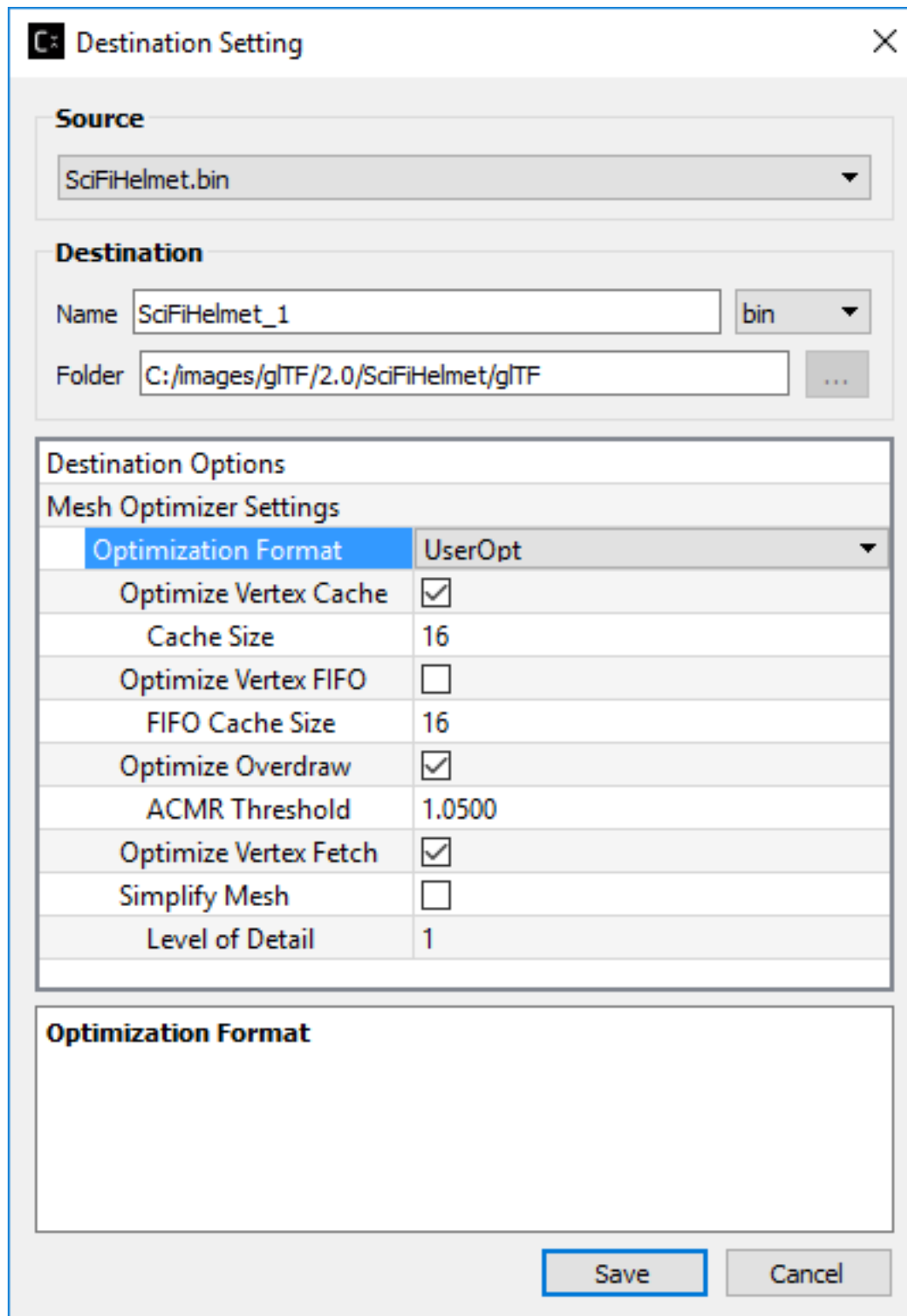
Note that you can add multiple 2nd level output node to the Project Explorer tree.

After that, expand the 2nd level output node (SciFiHelmet_1.gltf) by clicking on the right arrow next to the node and you will see the clickable "Add destination settings. . . ", click on "Add destination settings. . . ", Destination Option window will pop up and select a "Source" which are mesh (it will be a .bin file for glTF model and .obj file itself for obj model). The Destination Option window will be shown as below:
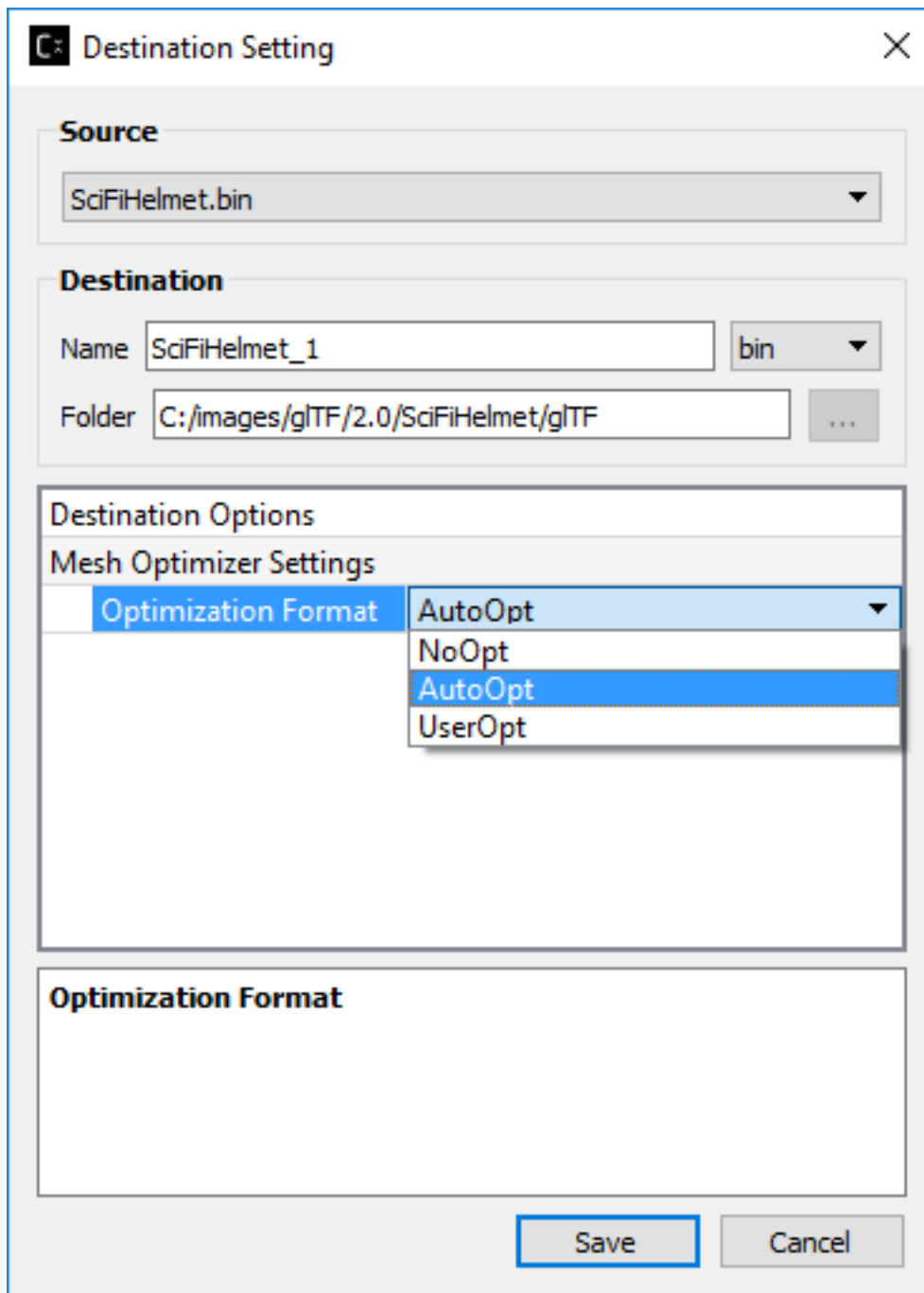
By default, "AutoOpt" is selected, which will perform the best optimization setting for the Model, you can override this setting by selecting "UserOpt" and set accordingly as described below.

Mesh Optimizer Settings for "UserOpt" selection

Optimize Vertex Cache

Optimize the cache utilization with LRU (least recently use) cache replacement policy.

Cache Size

Specifies the hardware vertex cache size for vertex cache optimization. This cache size refers to GPU built-in fixed size cache that store transformed vertices.

Optimize Vertex FIFO

Optimize the cache utilization with FIFO (first in first out) cache replacement policy.

FIFO Cache Size

Specifies the hardware vertex cache size for FIFO vertex cache optimization. This cache size refers to GPU built-in fixed size cache that store transformed vertices.

Optimize Overdraw

Reduce overdraw by reorder the triangles to render possible occludes first. Recommended to perform overdraw after vertex optimization if optimize vertex cache is checked. Overdraw optimization tries to maintain a balance with vertex cache optimization using the input ACMR Threshold.

ACMR Threshold

Average Cache Miss Ratio = #transformed vertices/#triangles (lower mean better vertex cache optimization). This is used for overdraw optimization process to make sure the overdraw optimization does not compromise vertex cache optimization. By default, it is set t0 1.05 (means resulting ratio at most 5% worse). Set to 3 to force overdraw optimization perform sorting on all triangles.

Optimize Vertex Fetch

Reduce overfetch from vertex buffer. This process will be performed after optimizing overdraw if optimize overdraw is enabled.

Simplify Mesh

Simplify the mesh by using 70% of the original index count and perform edge collapse algorithm using quadric and quadric error calculation. The target index count depends on the level of detail set by the user.

Level of Detail

Used in simplify mesh. Each level will use 70% of the index count on previous level. The higher the level, the less detail appear on the resulted mesh.

Mesh Compression Settings selection

As of v4.2 this option has been disabled.

When selected it will perform mesh compression using Draco encoder lib.
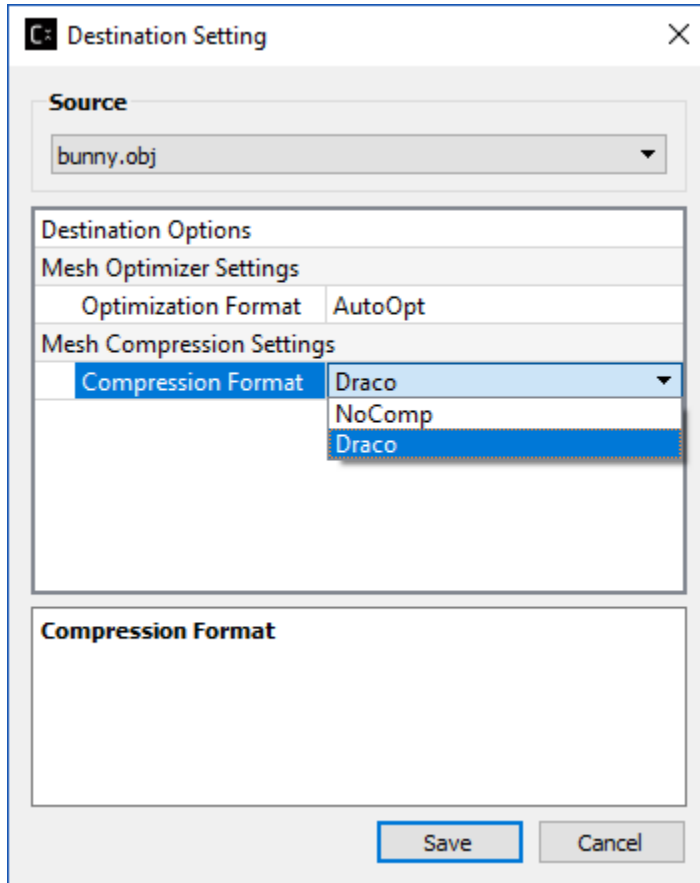
Click "Save" after completing the mesh optimizer and/or mesh compression settings. You will see a 3rd level destination settings node added to the Project Explorer as shown below:
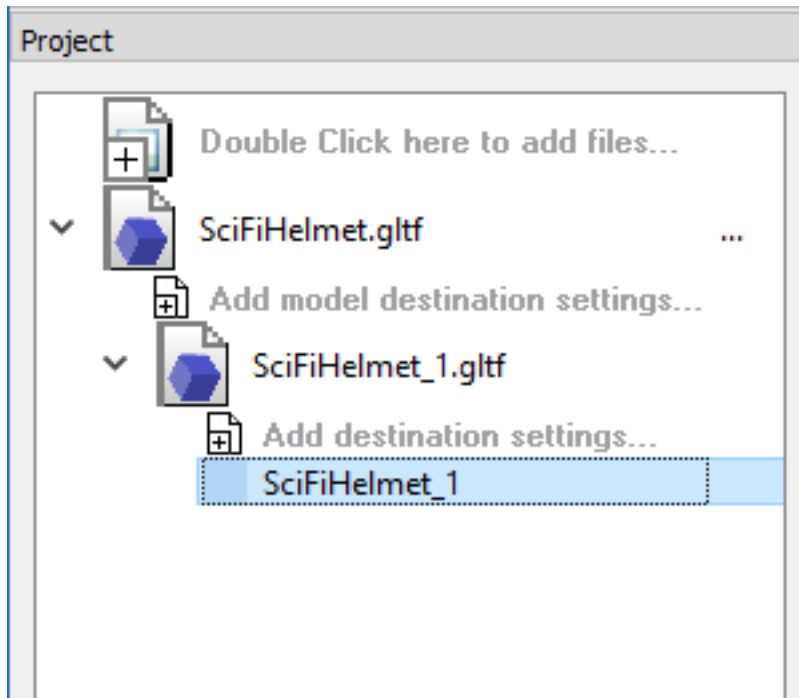
Right click on the 3rd level destination settings node and select "Process <node name>" as shown below:
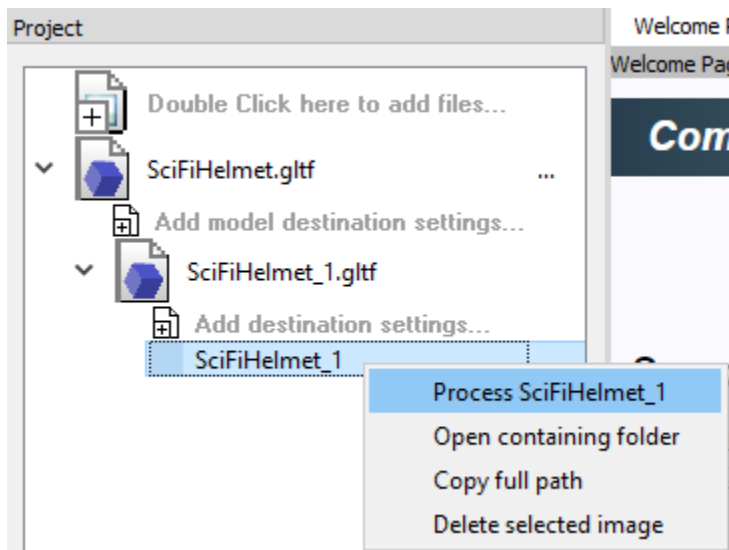
A green circle will appear beside the destination settings node after mesh optimization and/or compression completed.

Analyzing Mesh

Mesh analysis can be performed on the original model file as well as the resulted model file (1st and 2nd level items in the tree view of Project Explorer). Only .gltf and .obj files are supported for mesh analysis.

After adding the model file to the Project Explorer, right click on the file and select "Analyse Mesh Data" from the context menu:
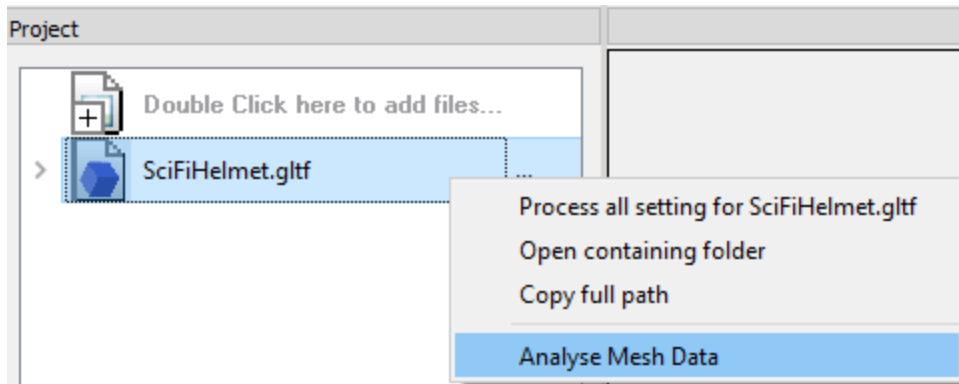
A 3D Mesh Analysis window will pop up as shown below:

3D Mesh Analysis

☑ Analyse Vertex Cache

Vertex Cache Size: `16`

Warp Size (# of threads): `64`

Primitive(Triangle) Buffer Size: `128`

☑ Analyse Vertex Fetch

Cache Line Buffer Size(bytes): `64`

Total Cache Buffer Size(bytes): `131072`

☑ Analyse Overdraw

Orthographic X and Y View Max Value: `256`

| Run Mesh Analysis | Cancel |
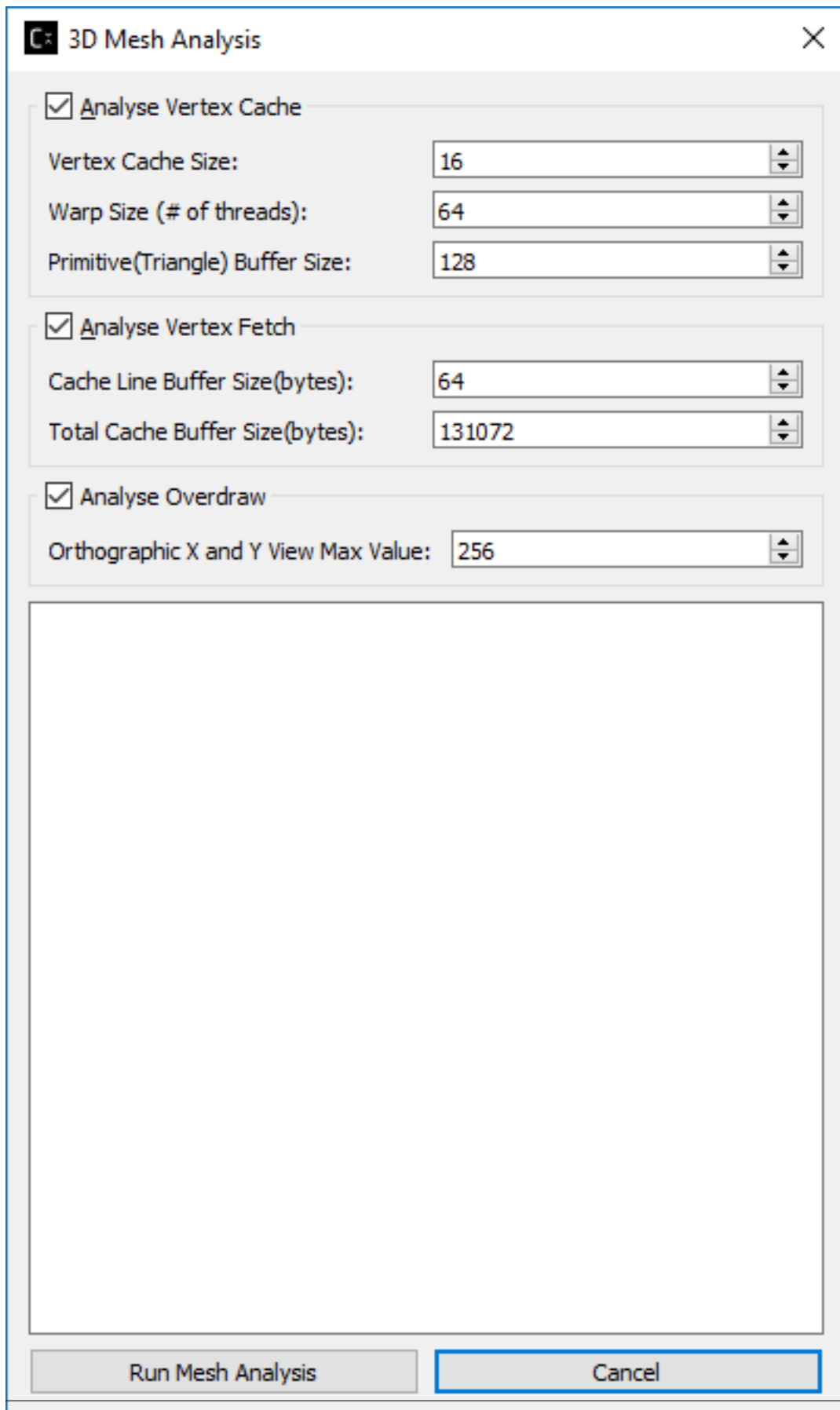
Analysis Vertex Cache

Vertex Cache Size

This is used to analyze vertex cache optimization. The vertex cache should be set to match mesh optimizer vertex cache size. Usually range between 16-32.

Warp Size (# of threads)

Number of threads per wave front group in GPU scheduling. 64 by default.

Primitive Buffer Size

Triangle group size. Default is set to 128.

Analysis Vertex Fetch

Cache Line Buffer Size(bytes)

This is the cache line buffer size which can be retrieved by running clinfo.exe in the window system. Default is set to 64.

Total Cache Buffer Size(bytes)

This is the total cache buffer size which can be retrieved by running clinfo.exe in the window system. Default is set to 128*1024.

Analysis Overdraw

Orthographic X and Y View Max Value

Overdraw statistic is calculated from different orthographic camera view start from 0. This set the maximum of X and Y viewport. Default is set to 256.

After done setting the desired mesh analysis settings, click "Run Mesh Analysis", if you run mesh analysis from the source model file (1st level tree item), the window will be updated with text output as shown below:

Cx 3D Mesh Analysis   ✕

☑ Analyse Vertex Cache

Vertex Cache Size:	16
Warp Size (# of threads):	64
Primitive(Triangle) Buffer Size:	128

☑ Analyse Vertex Fetch

Cache Line Buffer Size(bytes):	64
Total Cache Buffer Size(bytes):	131072

☑ Analyse Overdraw

Orthographic X and Y View Max Value:	256

==== Analysis Result for SciFiHelmet.gltf ====

==== Vertex Cache Analysis ====
Transformed vertices: 70074
Total vertices: 70074
ACMR (Average Cache Miss Ratio) : 3
ATVR (Average Transformed Vertices Ratio): 1
==== Vertex Cache Analysis Done ====

==== Vertex Fetch Analysis ====
Fetched vertices in Bytes: 2242368
Overfetch(fetched bytes/vertex buffer size): 1
==== Vertex Fetch Analysis Done ====

==== Overdraw Analysis ====
#Pixels rendered: 229217
#Pixels covered: 190602
Overdraw (pixels rendered/pixels covered): 1.20259
==== Overdraw Analysis Done ====

==== Analysis Complete ====

Run Mesh Analysis   Cancel

If you run mesh analysis from the resulted model file (2nd level tree item), the window is updated with both resulted model file analysis result as well as the original model file analysis result as shown below:

C⋮ 3D Mesh Analysis ✕

☑ Analyse Vertex Cache

Vertex Cache Size: `16` ⬍

Warp Size (# of threads): `64` ⬍

Primitive(Triangle) Buffer Size: `128` ⬍

☑ Analyse Vertex Fetch

Cache Line Buffer Size(bytes): `64` ⬍

Total Cache Buffer Size(bytes): `131072` ⬍

☑ Analyse Overdraw

Orthographic X and Y View Max Value: `256` ⬍

==== **Analysis Result for SciFiHelmet_1.gltf** ====

==== Vertex Cache Analysis ====
Transformed vertices: 70074
Total vertices: 70074
ACMR (Average Cache Miss Ratio) : 3
ATVR (Average Transformed Vertices Ratio): 1
==== Vertex Cache Analysis Done ====

==== Vertex Fetch Analysis ====
Fetched vertices in Bytes: 2242368
Overfetch(fetched bytes/vertex buffer size): 1
==== Vertex Fetch Analysis Done ====

==== Overdraw Analysis ====
#Pixels rendered: 205640
#Pixels covered: 190602
Overdraw (pixels rendered/pixels covered): 1.0789
==== Overdraw Analysis Done ====

==== **Analysis Complete** ====

==== **Analysis Result for SciFiHelmet.gltf** ====

==== Vertex Cache Analysis ====
Transformed vertices: 70074
Total vertices: 70074
ACMR (Average Cache Miss Ratio) : 3
ATVR (Average Transformed Vertices Ratio): 1
==== Vertex Cache Analysis Done ====

==== Vertex Fetch Analysis ====
Fetched vertices in Bytes: 2242368

This is essential for users to compare between original and processed model file mesh analysis.

Mesh Statistic

ACMR (Average Cache Miss Ratio)

ACMR = #transformed vertices/#triangles. The average number of cache miss per triangle is 0.5 in optimum case and 3.0 in worst case. Lower mean better vertex cache optimization.

ATVR (Average Transform Vertices Ratio)

ATVR = #transformed vertices / vertex count. The optimum case is 1.0, worst case is 6.0. Lower mean better vertex cache optimization.

Overdraw

Overdraw = #pixels shaded/total pixels covered. The best case is 1.0 (each pixel is shaded once)

Overfetch

Overfetch = #bytes read from vertex buffer/total # bytes in vertex buffer. The best case is 1.0 (each byte is fetched once)

Note: As mesh processing is model dependent. These mesh statistics shown in mesh analysis can be used as a guideline to produce better mesh using the *Mesh Optimizer Settings*.

3.2.5 Mipmap Generation

Mipmaps can be generated for one or more source images in the Project Explorer by selecting the image(s) and clicking

the "Generate Mipmaps" button in the Application Toolbar.

This will cause a dialog box to appear with some parameters related to mipmap generation.
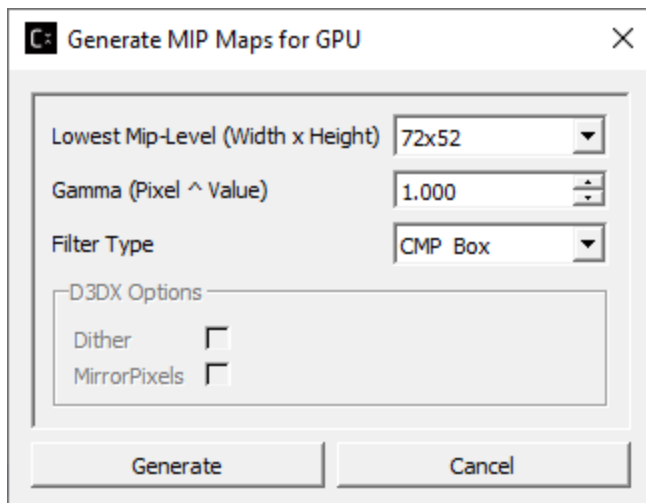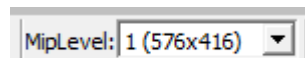
Select the minimum mipmap level size you desire from the drop-down list and press the "Generate" button.

The available mipmap sizes changes based on a few different variables. By default, all supported sizes for the current selected image will be display. If you have chosen to generate mipmaps using the GPU in the Application Options then only sizes divisible by 4 will be shown. Finally, if multiple images are selected a set of "standard" sizes will be shown and you should choose the one closest to your desired goal.
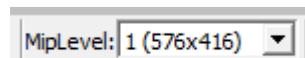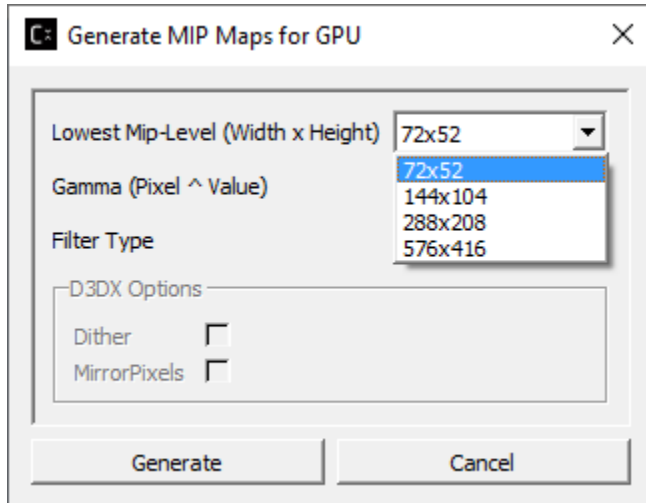
When mipmap generation completes, will appear in Image View toolbar. Click on the little triangle to expand the drop down list.
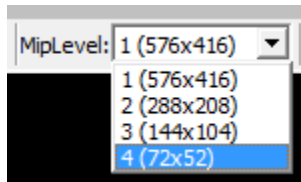
You can select the view of each mipmap level from the list.

For example, selecting the level 3 will update the current image view as shown below

---
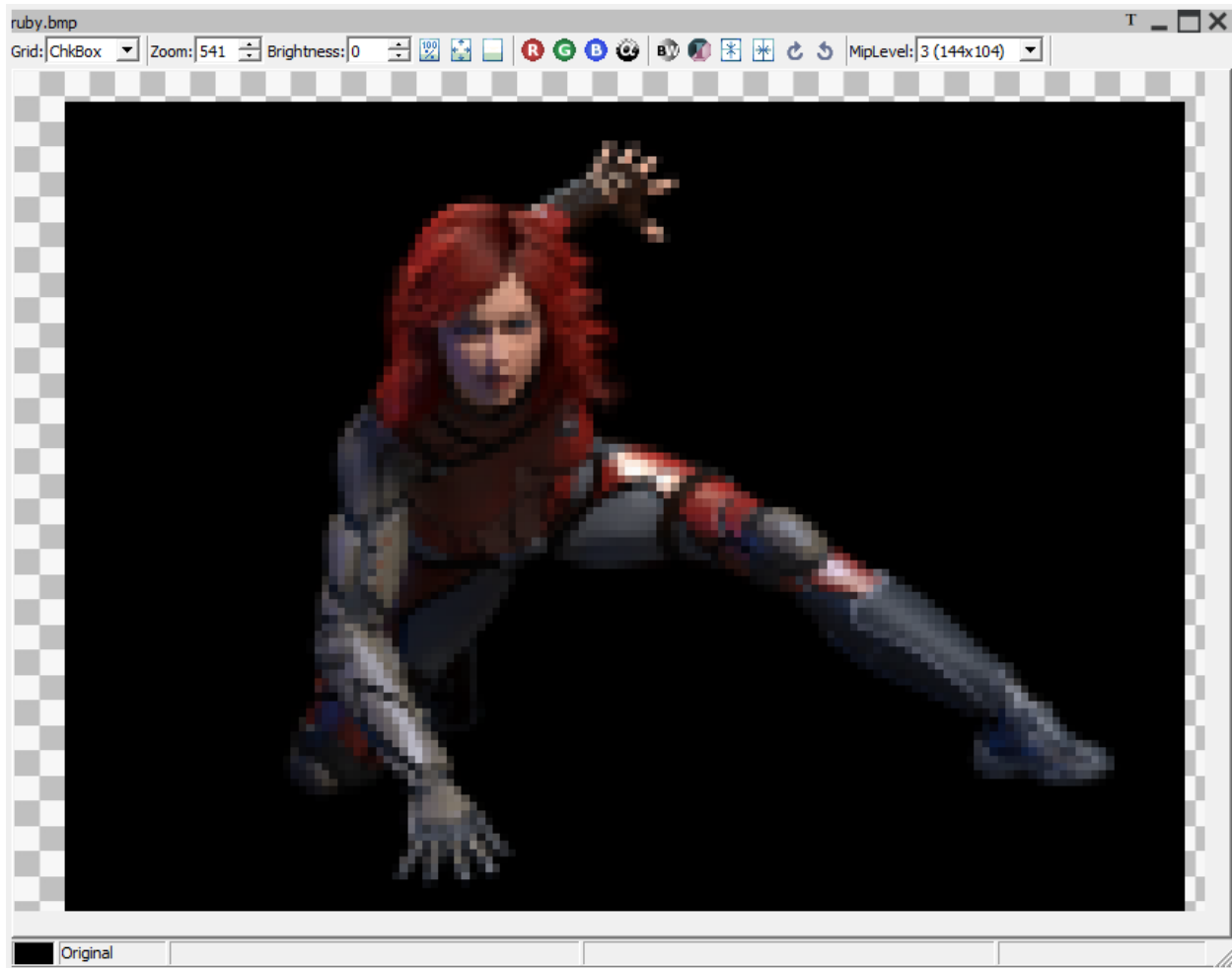
Image View of Ruby BMP file at Mip Level 3

You can always re-generate mipmaps for the same image by repeating all the steps mentioned in this section.

3.2.6 Mipmap Filter Support Using either GPU or CPU

The GUI applications "Generate Mipmaps" option can be used on a variety of image formats to produce mipmaps filtered for optimal GPU or CPU use.

Options for using a Compressonator Box filter or DirectX® based filters are provided.
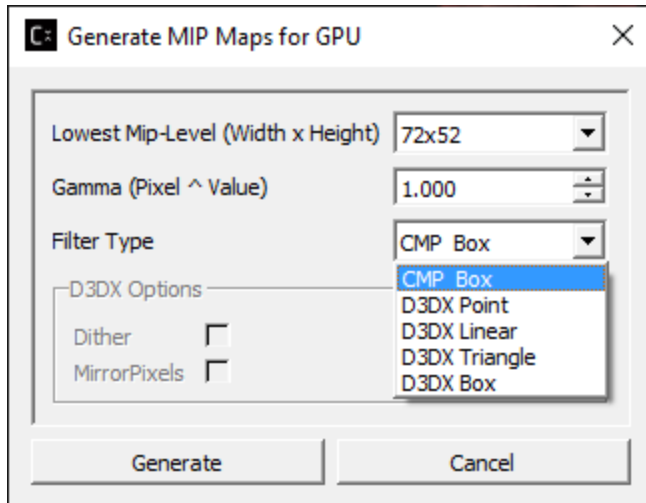
3.2.7 Applying Gamma to Mipmaps

Gamma correction can be applied to the image pixels after mipmap generation by changing the default 1.0 value higher (darken the image) or lower (brighten the image).

3.2.8 GPU Based Compression and Mipmap generation

An alternative option to generate mipmap levels is provided using the application setting window. A feature to encode textures using just the GPU hardware and its driver components is provided. The encoding process uses no user application shader code to process the textures. Both the GUI and Command-line tools can run BCn GPU extensions provided by most GPU vendors and can be used to evaluate the quality and performance of encoded images using the image views and analysis setting.

Just enable the GUI "Application Settings" options to set the encoding with GPU and optionally set generating GPU based mipmaps as shown:
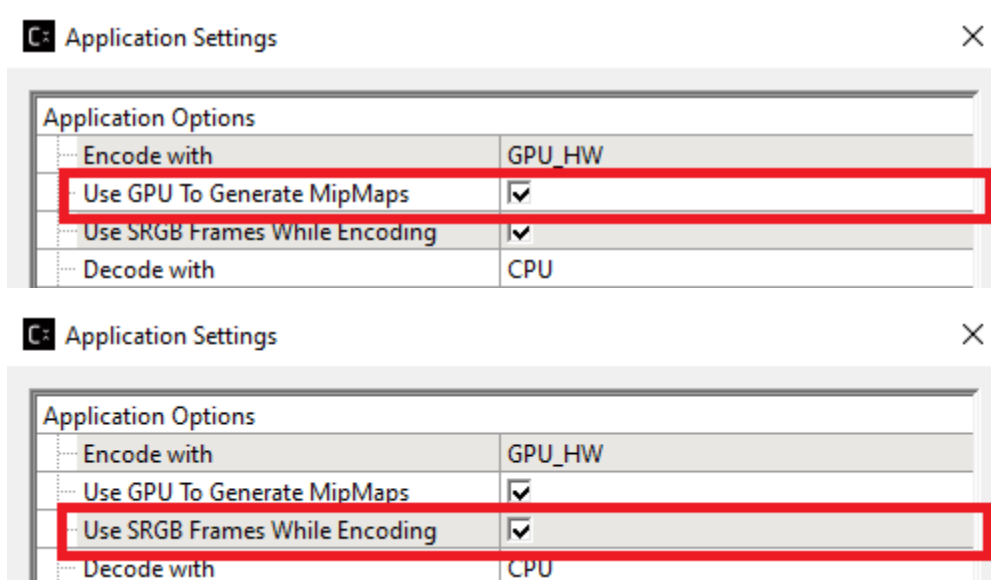
Once the images are processed users can view the quality of the resulting images with the applications image view PSNR feature,

3.2.9 Cube Maps

Cube Maps can be compressed with or without mipmap levels. Only a limited set of texture types (RGBA_8888 and RGBA_F16) are currently supported in DDS and KTX file formats. Compressing, generating mipmap levels, and viewing cube maps uses the same process as image textures. Just place the file onto the Project Explorer and process them as required.

A new notation is used for the cube faces labeled as "Frames" for each cube face. For cube mapped files maximum frames is set to 6. Support for volumetric texture files is been reviewed and the frames limit will be expanded as needed.

When textures are added to the Project Explorer, the properties view will now display the type of texture as either a 2D or Cube Map, The Depth field is used for the frames size, the Depth field will also be used to indicate the z component of a 3D image (These notations may change in future revisions).
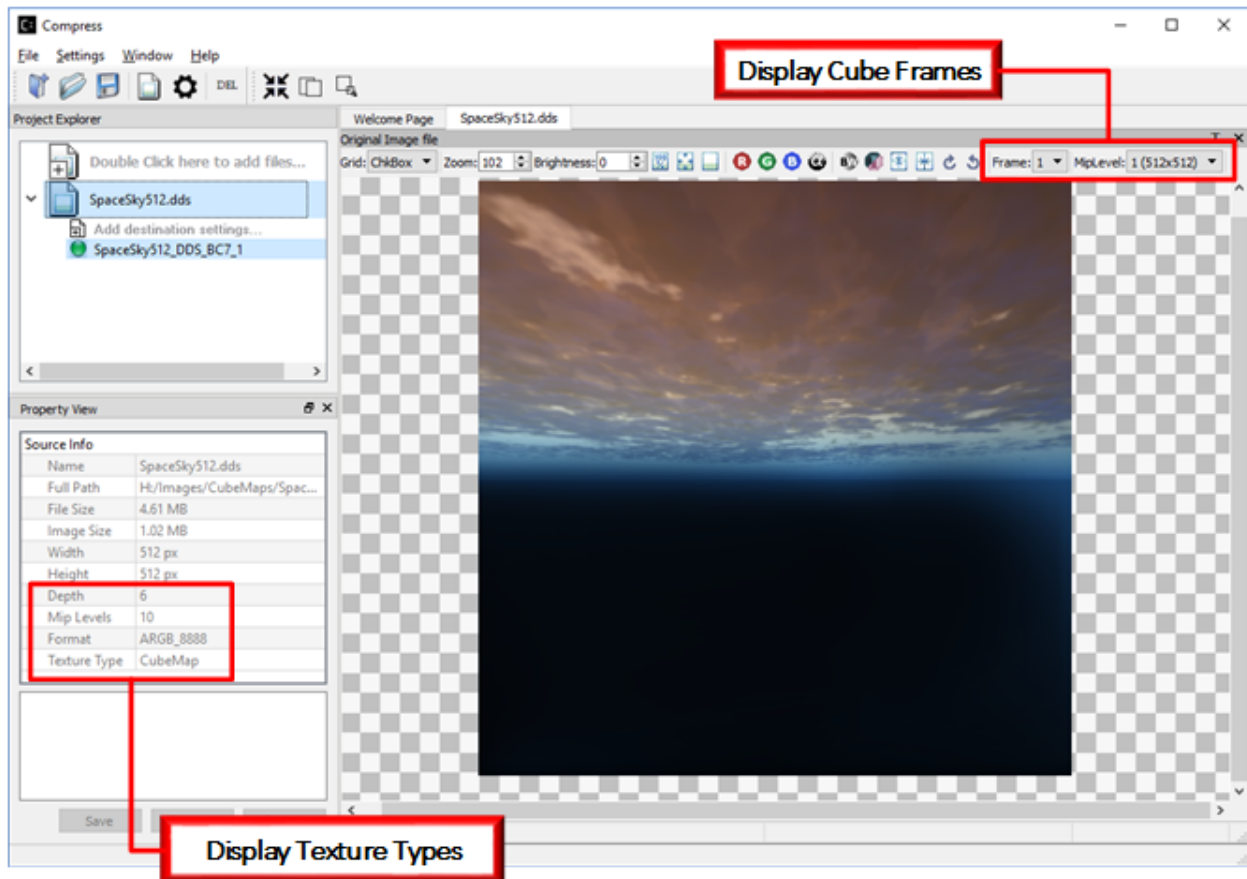
3.2.10 View Image Difference

To view the difference between a processed image (Ruby_bmp_1) and original image (Ruby_bmp), right mouse button click on Ruby_bmp_1 and select View Image Diff from the context menu or select the View Image Diff Icon on the

tool bar
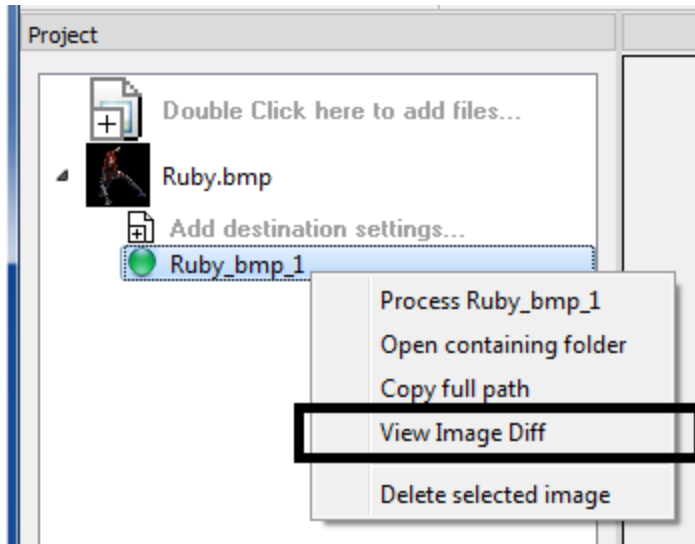
Mouse right mouse button click over Ruby_bmp_1 showing Context menu

You will now see a comparison of the original image with the compressed image
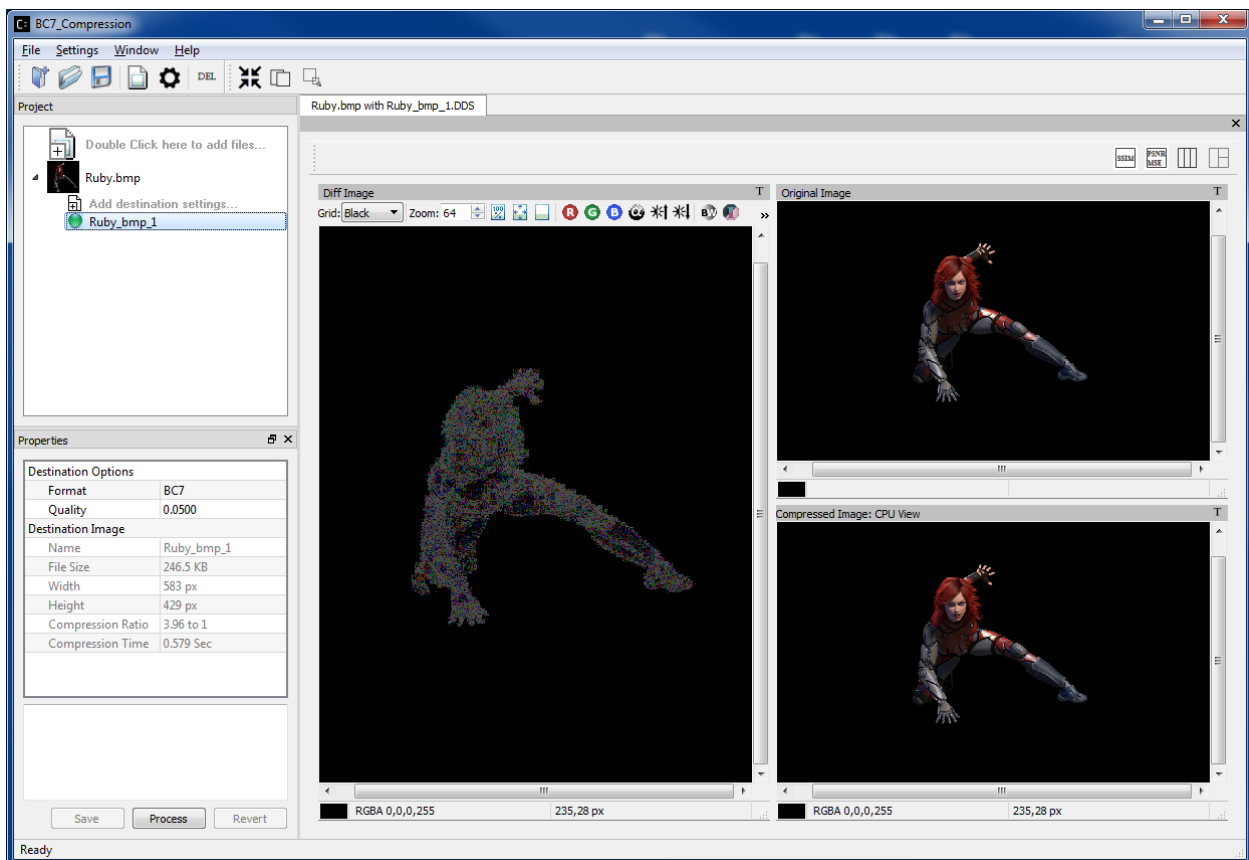
Image Difference view
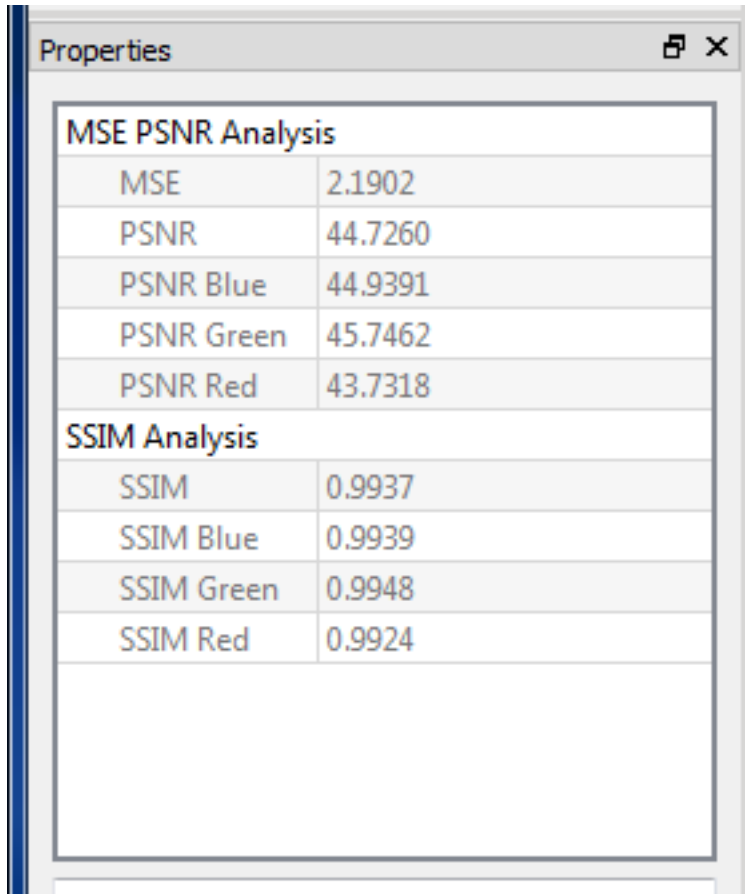
Note: The windows in this view are not movable.

3.2.11 Analyzing Compressed Images

After clicking View Image diff, you can run analysis on the images that show various statistics such as MSE, PSNR

and Similarity Indices (SSIM) by selecting ⬚ on the top right corner of the image diff view.

When analysis process completed, the statistics result will be shown on the Property View

Properties		⧉ ✕
MSE PSNR Analysis		
MSE	2.1902	
PSNR	44.7260	
PSNR Blue	44.9391	
PSNR Green	45.7462	
PSNR Red	43.7318	
SSIM Analysis		
SSIM	0.9937	
SSIM Blue	0.9939	
SSIM Green	0.9948	
SSIM Red	0.9924	

Build from GitHub Sources

The simplest way to get Compressonator is to use any of the pre-built binaries available on the GPUOpen-Tools GitHub page. These include the command line tool, GUI application, and binaires for installing the SDK for developers.

This page will serve as a guide for the more advanced user who desires the flexibility of building Compressonator directly from the source code.

For either case, whether you want pre-built binaries or to download the source for the latest release of Compressonator, you can use the following link:

https://github.com/GPUOpen-Tools/Compressonator/releases

While the releases page should provide you with a stable version of Compressonator to use, there are occasionally small updates made between official releases that fix minor bugs. So if you need the absolute latest public source code, you can run the following command:

```
git clone --recursive https://github.com/GPUOpen-Tools/Compressonator.git
```

4.1 Prerequisites

There are a few dependencies that must be installed before Compressonator can be built. The following are applicable for both Windows and Linux builds:

- CMake 3.15 or above
- Vulkan SDK version 1.2.141.2 or above is required
- Python 3.6 or above
- Qt 5.12.6 is recommended
- OpenCV 4.2.0 is recommended

Additionally, on Windows there are a couple of extra dependencies you will want to ensure are installed. The first is the Windows 10 SDK version 10.0.19041.0 or later. You can typically download this through the Visual Studio Installer, but it can also be found via the Windows SDK Archive. Also, you will want to make sure that you are using

toolset version 142 when compiling the code in Visual Studio. This is the default for Visual Studio 2019, but for newer versions you will need to install the correct toolset.

4.2 Building the SDK and Other Libraries

There are Visual Studio 19 project files provided in the "compressonator/vs2019" folder that can be used to build the SDK (also called cmp_compressonatorlib), the Core, and the Framework.

There is a batch script provided in the "compressonator/scripts" folder named "windows_build_sdk.bat" if you'd prefer to run an automated script to build all the SDK libraries. Be warned that this script builds all variations of the SDK libraries (in both Release and Debug modes), so it can take quite a while to fully build.

The script can be called from the root Compressonator folder like:

```
call scripts/windows_build_sdk.bat
```

4.3 Building the GUI and CLI Applications

Similar to the SDK libraries, there is a simple script you can call on Windows that will setup the environment and start the builds for the Compressonator applications. It does make some assumptions about the specific versions of Qt and the Vulkan SDK that are installed, so it might not work without some tweaking. The batch script is located at "scripts/windows_build_apps.bat".

If you'd prefer more customization, or wish to build from Linux, the steps to generate the projects and build the apps will be detailed in the remainder of this section.

4.3.1 Preparing the Environment

NOTE: On Linux you will want to start by calling the "initsetup_ubuntu.sh" script in the "scripts" folder to download and install any other dependencies you might be missing.

The first step is to execute the "fetch_dependencies.py" script located in the "scripts" folder, using a command like:

```
python fetch_dependencies.py
```

This will download all of the external libraries used by Compressonator into a "common" folder at the same level as the root Compressonator folder.

4.3.2 Generating Project Files

The next step is to use CMake to generate the project files. But before you can do that a few environment variables need to be set.

There are a few libraries that Compressonator expects the user to install on their own and then tell Compressonator where to find them. Those libraries are: the Vulkan SDK, Qt5, and OpenCV. To let Compressonator know where to find these libraries we use environment variables. The variables are **VULKAN_DIR**, **QT_DIR**, and **OPENCV_DIR** respectively.

They can be set by doing running commands like the following:

Windows

```
set VULKAN_DIR=C:\VulkanSDK\1.2.141.2\
set QT_DIR=C:\Qt\Qt5.12.6\5.12.6\msvc2017_64\
set OPENCV_DIR=C:\opencv\
```

Linux

```
export VULKAN_DIR=/opt/VulkanSDK/1.2.141.2/
export QT_DIR=/opt/Qt/Qt5.9.2/5.9.2/gcc_64/
export OPENCV_DIR=/opt/opencv/
```

With all of that said, you are now ready to run CMake and generate the project files. An example of the CMake command to run on both Windows and Linux can be found below (run from the root compressonator folder that contains the CMakeLists.txt file):

Windows

```
cmake -G "Visual Studio 16 2019" .
```

If you are instead using Visual Studio 17 2022 (or newer) you will need to change the value passed to the -G option, you will also need to explicitly set the toolset version used to v142 via the -T option ("-T v142").

Linux

```
cmake .
```

4.3.3 Building Compressonator

Assuming the CMake command finished without incident, everything is now set up to build Compressonator!

If you are on Windows you can open the "compressonator.sln" file and build using Visual Studio. If you'd rather use the command line, or are running Linux, the commands to build would look something like:

Windows

```
msbuild /m:4 /t:build /p:Configuration=release /p:Platform=x64 compressonator.sln
```

Linux

```
make
```

4.4 Optional Build Settings

Compressonator provides many optional flags you can use to customize your build when running the CMake command. You can check the root CMakeLists.txt file for everything that exists, but in this section we will point out a few that might be of interest.

Some of these will let you choose to build only certain parts of the project, while others will allow you to reenable features that are no longer officially supported and are turned off by default.

- **OPTION_ENABLE_ALL_APPS** Allows you to enable or disable building of the CLI and GUI applications. This is ON by default.
- **OPTION_BUILD_APPS_CMP_CLI** Enable only the CLI application for building.
- **OPTION_BUILD_APPS_CMP_GUI** Enable only the GUI application for building.

- **OPTION_BUILD_DRACO** Enable using the Draco library for compressing and decompressing 3D meshes. This is OFF by default.

- **OPTION_BUILD_ASTC** Enable the ASTC codec. This is OFF by default.

4.5 Building the Documentation

The documentation is written using the reStructuredText markup syntax. There are batch files and scripts provided that will run commands to build the documentation into HTML pages for easy viewing. These scripts use Sphinx to build the documentation, so you must install that first before you can use them.

Once Sphinx is installed, you can run the following from the root Compressonator folder for Windows:

```
call scripts/windows_build_docs.bat
```

For Linux users you need to call the make files directly in the "compressonator/docs/" folder, something like:

```
set -x
cd compressonator/docs
make -j 4 clean
make -j 4 html
```

Analysis

Compressonator CLI and GUI applications provides options for analysis and comparison between original and processed images.

5.1 Test Analysis Log Features And File Filters

Ref: CLI Analysis Log File:

Captures details of the source and destination files along with statistical data on performance and quality into a text file "process_results.txt"

Example: **CompressonatorCLI.exe -log <source directory> <destination directory>**

5.2 CompressonatorCLI Analysis

Compressonator CLI has a command line option to run Image Analysis on two sample images of the same dimension and format using the option -analysis

Example:

Step 1: Do compression on a sample source image, say we used BC7 on a source file Ruby.dds and the destination file Ruby_BC7.dds

CompressonatorCLI.exe -fd BC7 ruby.dds ruby_bc7.dds

Step 2: Check the result of the compressed image with the original

CompressonatorCLI.exe -analysis ruby.dds ruby_bc7.dds

The analysis results is saved to file Analysis_Result.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ANALYSIS>
    <ruby_ruby_bc7>
        <MSE>0.81335721644469494</MSE>
        <SSIM>0.99769089871409922</SSIM>
        <SSIM_BLUE>0.99735912437459595</SSIM_BLUE>
        <SSIM_GREEN>0.9974541763291831</SSIM_GREEN>
        <SSIM_RED>0.99825939543851883</SSIM_RED>
        <PSNR>49.027990366039383</PSNR>
        <PSNR_BLUE>48.733856839394129</PSNR_BLUE>
        <PSNR_GREEN>49.702877166369824</PSNR_GREEN>
        <PSNR_RED>48.71854158766061</PSNR_RED>
    </ruby_ruby_bc7>
</ANALYSIS>
```

Notice the markup label below <ANALYSIS> contains the source file name "ruby" and destination file name "ruby_bc7" appended by an underscore "ruby_ruby_bc7"

SSIM is the calculated average of RGB Channels. A simple file parser can then be used to run SSIM and other Matrix analysis for test automation.

if you wish to generate diff image file, run

"CompressonatorCLI.exe -analysis -diff_image ruby.dds ruby_bc7.dds" in Step 2.

A image diff (i.e. ruby_diff.bmp) will be generated. Please use the Compressonator.exe (UI app) to view the diff bmp file with adjusted brightness levels.

5.3 Analysis For Images In Folders

1. Process a folder of images and performa analysis between images inside the source folder and destination folder

CompressonatorCLI.exe -fd <Codec format> <source directory> <destination directory>

Example: `CompressonatorCLI.exe -fd BC1 -log ./images  ./results`

2. Generate analysis report between a source image and a processed image

CompressonatorCLI.exe -analysis <source image file> <resulted image file>

For example: `CompressonatorCLI.exe -analysis Boat.png result.dds`

This option will generate an Analysis_Result.xml report file which contain the SSIM, PSNR and MSE values between the original and processed textures.

3. Generate a difference image between a source image and a processed image

CompressonatorCLI.exe -diff_image <source image file> <resulted image file>

For example: `CompressonatorCLI.exe -diff_image Boat.png result.dds`

This option will generate difference between 2 images with same size. A result_diff.bmp file will be generated. Please use compressonator GUI to increase the contrast to view the diff pixels.

4. Print image properties of an image file.

CompressonatorCLI.exe -imageprops <image file>

For example: `CompressonatorCLI.exe -imageprops Boat.png`

This option will print file name, path, size, image width, height, mip levels and format.

5.4 Analysis Views

A new table view is provided for users to view test analysis results for performance, PSNR and SSIM analysis during and after processing of image textures. Previously in v3.1 user would have to run "Image Difference Views" on each processed texture to view PSNR and SSIM results, which ends up been very time consuming for large number of images. An amalgamated results view is now available for users to collect and review results for processed images. To enable this feature, select the "Show MSR PSNR SSIM Results" and optionally "Show Analysis Results Table" in the "Applications Settings" menu as shown:
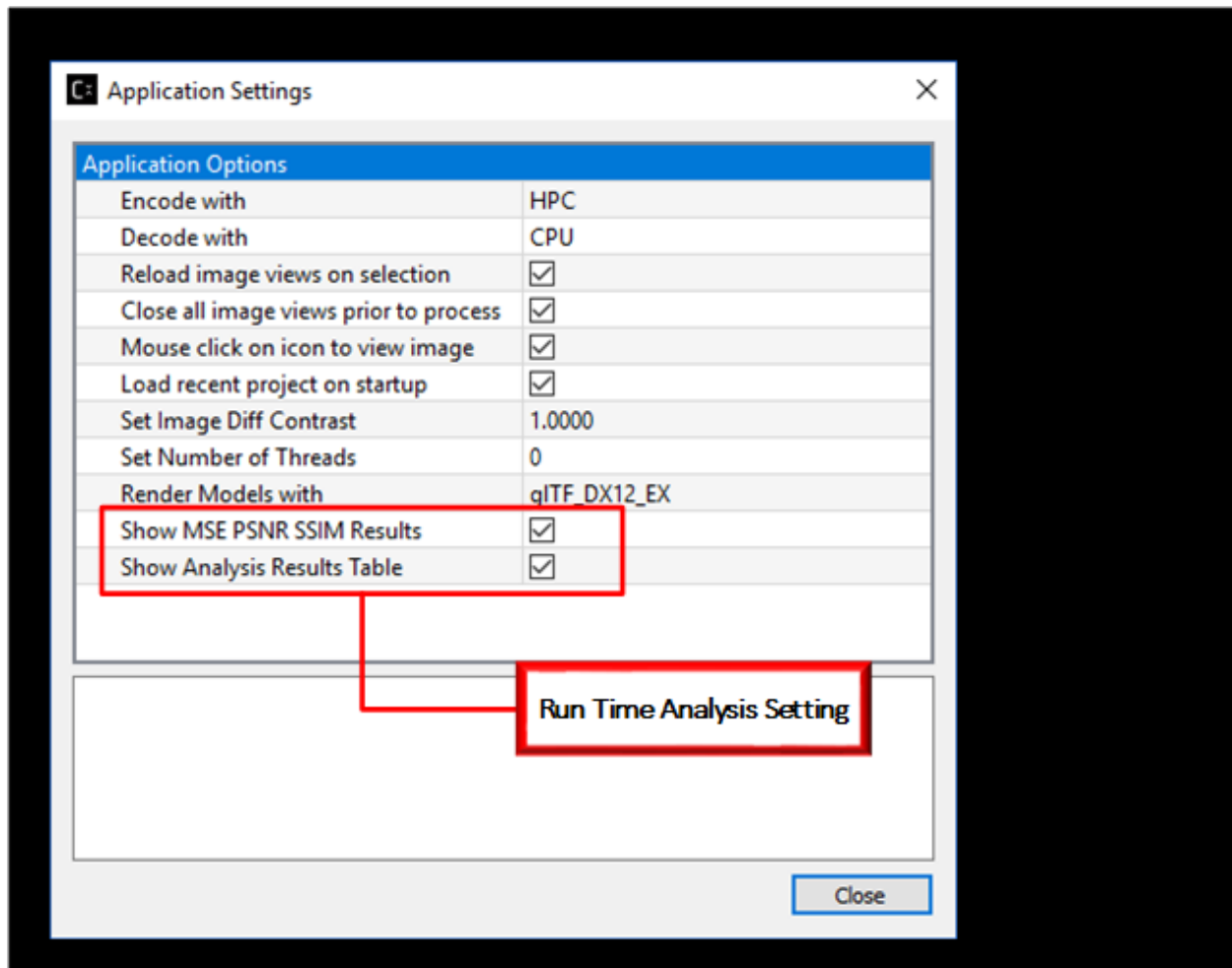
Show MSR PSNR SSIM Results

Enables the analysis results feature. Each processed item will display a MSE, PSNR and SSIM as it is been processed on the output window. A summary of the average processing time PSNR and SSIM is displayed once all processing is completed (MSR average results is not collected as its already included in the PSNR calculations). It also enables the "Show Analysis Results Table" for selection. This selection is active only when "Show MSR PSNR SSIM Results" is enabled, else the table will not be shown after processing

Output   **x**

```
Processing: kodim01_HPC
CompressonatorCLI.exe   H:/Images/DDS/kodim01.DDS   H:/Images/DDS/results/kodim01_HPC.DDS   -fd BC7  -EncodeWith HPC  -Quality 1.0000
Total time taken (includes file I/O): 8.372 seconds
MSE 1.88 PSRN 45.4 SSIM 0.9968
Done Processing
Processing: kodim02_HPC
CompressonatorCLI.exe   H:/Images/DDS/kodim02.DDS   H:/Images/DDS/results/kodim02_HPC.DDS   -fd BC7  -EncodeWith HPC  -Quality 1.0000
Total time taken (includes file I/O): 9.538 seconds
MSE 1.43 PSRN 46.6 SSIM 0.9926
Done Processing
Processing: ruby_HPC
CompressonatorCLI.exe   H:/Images/DDS/ruby.dds   H:/Images/DDS/results/ruby_HPC.DDS   -fd BC7  -EncodeWith HPC  -Quality 1.0000
Total time taken (includes file I/O): 1.300 seconds
MSE 0.64 PSRN 50.0 SSIM 0.9983
Done Processing
Processed in 20.3 seconds
====== Compress: 3 succeeded, 0 failed, Time 6.39, PSNR 47.3, SSIM 0.9959 ======
```

Show Analysis Results Table Displays this table after any image is been processed (TIME is in Seconds, PSRN is in dB and SSIM is an index in range of 0...1)

Analysis Encode with DXC   —   □   ✕

	File Path	File Name	Quality	KPerf(ms)	MTx/s	Time(s)	PSNR(dB)	SSIM
1	C:/Perforce/main/Comp...	ruby_BMP_BC1_1	0.05	51.036	0.314	0.924	38.4	0.9848
2		Average DXC				0.924	38.4	0.9848

When processing images the table view will be displayed and can be used to gather results and analysis data for reports, by simply selecting the required rows and columns in the table then copy (Ctrl+C) and pasting (Ctrl+V) the table view.

KPerf(ms) (Beta Feature)

This estimates the time it takes to process 1000 (4x4 pixel) blocks, using the current encoder and GPU setup. where Perf(ms) is the time it took to process a single block of 16 pixels in milliseconds.

HPC performance monitoring uses CPU timers. while OCL & DXC uses GPU performance query timers. For CPU based encoding KPerf & MTx is not measured.

MTx/s (Beta Feature)

This is a measurement of the time it takes to process 1 Million texels in a second.

Time(s)

CPU performance based timing, that measures the overall end to end time it took for the image to be processed. It includes device setup, loading image to GPU, receiving the image from GPU and file IO.

Please note the following

- Analysis is not supported for processes that are: Lossless, HDR to LDR or LDR to HDR
- The table view is re-sizable by mouse clicking on and dragging the table edges and dividers.
- The resized view and table position are not saved and is reset when the GUI application is restarted.
- Table Header labels are not copied.
- Table content is cleared when any of the image textures are been reprocessed.
- When enabled, the table will remain as a topmost windows view.
- During processing the table content is been updated.

- When deselecting the analysis table view from the "Applications Settings" menu, the table will still remain visible, it is cleared only when a image is been processed.

Revision History

For the latest documentation, please refer to: http://compressonator.readthedocs.io/en/latest/

6.1 V4.4

July 2023

Features

- SIMD Support in BC1 Encoding

    - Added new code paths for SSE4, AVX2, and AVX-512 encoding in Compressonator Core that are automatically chosen based on CPU support

    - New functions added to the core interface to allow users to manually enable/disable SIMD instructions if desired: **EnableSSE4()**, **EnableAVX2()**, **EnableAVX512()**, and **DisableSIMD()**

    - Another new function was added to allow users to query the SIMD instruction set extensions currently being used in Compressonator Core: **GetEnabledSIMDExtension()**

- Mipmap Generation Updates

    - New option called "FilterGamma" added to Compressonator CLI which allows users to apply gamma correction to mipmap levels after generation

    - Mipmap generation behaviour has been modified to only stop once all dimensions of the mipmap level are equal to or below the minimum size specified

- Miscellaneous Changes

    - The default location of build artifacts has changed from *build* to *build/bin*, and build scripts have been moved to *build*

    - A new CMake project for building only the SDK in various configurations has been created under the *build/sdk* folder

- The regular CMake build now requires three environment variables to be set that point to external dependency installations: **OPENCV_DIR**, **QT_DIR**, and **VULKAN_DIR** (more details can be found in the build section of the docs)

- 16-bit PNG support added to Compressonator Framework

- Removed implicit channel swizzling from all codec buffer classes, ensuring more consistent behaviour

- Renamed CMP_TestCore project to CMP_UnitTests and expanded the suite of unit tests

- Fixed compilation errors for BC6H core shader

- Dependency on DirectX 9 and DirectX 10 DLL files has been replaced with the use of DirectXTex

- Silent option in Compressonator CLI properly disables all output other than fatal errors.

- Edited documentation to fix typo for "GenGPUMipMaps" option

- Improved BC7 codec performance by keeping initialization data in memory rather than freeing it after each call to CMP_ConvertTexture

Known issues and limitations

- Compressing RG8 format images to BC5 results in black data in the output red channel

- Potential buffer overflow in RG8 Codec Buffer class when converting RG8 data to BC5

- Swizzling is not supported in all variations of the codec buffer classes

6.2 V4.3

January 2023

Features

- Brotli-G

  - New lossless compression format CMP_FORMAT_BROTLIG that can compress all types of data, not just textures

  - Command line support via CompressonatorCLI as format "BRLG"

  - Encoding is done using the CPU and decoding can be done by either the CPU or GPU

  - New Compressonator file format for compressed Brotli-G data, ".brlg"

- Mipmap Updates

  - Generating mipmaps in the GUI app will process every selected image, rather than stop after the first one

  - Fixed crash when trying to generate mipmaps for a file with existing mipmaps

  - Fixed crash when attempting to generate mipmaps for greyscale images through the GUI app

- Compressed to Compressed Format Transcoding

  - Supports BCn to BCn conversions

  - Works by using a temporary uncompressed format as an intermediate: compressed input → uncompressed → compressed output

- RGBA1010102 Format Support

  - Added new format: CMP_FORMAT_RGBA_1010102 which is a variation of the existing CMP_FORMAT_2101010 format

- – Added ability to load DDS files in the new format

- Option to Use Original File Names when Compressing

  - – Changed default behaviour of CompressonatorCLI so that it no longer "mangles" names, meaning it no longer adds extra text to the end of destination file names to identify the source file extension and destination format.

  - – When processing batches of files CompressonatorCLI will use the source file names as the destination, unless multiple sources share the same file name. In that case, it will mangle the file names of the repeated files.

  - – Added the flag "-UseMangledFileNames" for CompressonatorCLI that will revert back to the previous behaviour of mangled destination file names.

  - – A new application setting was added to CompressonatorGUI, "Use Original File Names", which will use the source's file name as the destination's file name for the first destination per source. This is enabled by default.

- BC6H Codec Improvements

  - – Better handling of edge cases like infinity and NaN values in the input data

  - – BC6H_SF preserves negative numbers through compression and decompression

- Mesh Optimization

  - – Saved results can no longer contain invalid GLTF data when the input lacks some properties, like animations

  - – Fixed handling of file paths so that paths with drive letters specified in them work as expected

  - – Improved error messages and error handling

- GUI Compression Consistency Improvement

  - – Changes made to the three main processing buttons: the button in the top bar, the button in the property view, and the right-click context menu. Their behaviour is now more consistent regardless of program state.

- Miscellaneous Changes/Improvements

  - – CMP_FORMAT_RGBA_32F is recognized as a valid format for compression

  - – Fixed BC4 compression with R8 input

  - – Various small improvements in how CompressonatorCLI handles files and folders, especially in batch processing

Known issues and limitations

- Transcoding decreases image quality.

- R8 images are currently loaded into the red channel of a 4 channel ARGB_8888 image. This does not affect the results of any compressions, but the GUI will continue to show ARGB_8888 instead of R_8.

- The GPU_HW encoding option produces corrupted results on some machines when encoding to BC4_S or BC5_S. There is also an issue with generating and compressing mipmaps using the GPU_HW option that results in the lower mipmap levels being completely blank on some machines.

- Brotli-G encoding might sometimes result in compressed files that are slightly larger than the source file, particularly for well compressed sources. This is a limitation similar to ZIP compression, where small files or well compressed images might not be able to be compressed very well.

- Brotli-G encoding treats all source files as merely binary data, so it will not be able to create mipmaps or do any other extra processing to images during encoding.

6.3 V4.2

July 2021

Features

- BC1 Quality Improvements
    - Added new refine steps to improve quality of images with mixed low and high-frequency content.
    - Boosted encoding performance by 2x with improved quality for smooth texture-mapped surfaces.
- BC7 HLSL Improvements
    - Bug fixes and improved overall quality of the shader.
- GUI Update
    - Added "Refine Steps" settings for BC1 image processing
    - Removed "Use Alpha" and "No Alpha" setting for BC1, Just set a threshold above 0 to enable BC1 punch through ranges
- SDK Updates
    - Removed framework.h file, replaced with compressonator.h
    - Added new kernel setting options for BCn codecs in CMP_Framework
    - New single header shader files added for CPU-GPU common API's and type definitions
    - Added FidelityFX types to common_defs.h to support programming FidelityFX shaders with CMP type definitions
    - Improved CompressonatorLib BC1 performance and quality
- CMake Build Updates
    - Added options to select build of external libs, SDK libs and applications
    - Build updated for CLI on Mac OS
    - OpenCV 4 supported
    - Improved compiler standard revision detection and extensions
    - Visual Studio 2019 support
    - Qt 5.15 support

Known issues and limitations

- BC1 set to high-quality settings or with refinement steps take longer time to process than prior versions, adjust quality-settings lower if better performance is required
- Fixed quality issues for BC7 HLSL for images with alpha content, the processing time is longer than the previous release
- Global Setting of Refine Steps is only valid for BC1, it will not work for any other format settings.

6.4 V4.1

November 2020

Features

- SNORM Support

    Signed channel support for BC4 and BC5

- GPU Encoding

    GPU Based Compression

- Mip Map Generation

    MIP Map Filter Support Using either GPU or CPU

    GPU Based MIP Map generation

- Image View

    PSNR Display Feature for GUI Image Views

- Test Analysis

    CSV File Update to Support Automation

- KTX2 File Support

    A KTX version 2 plugin has been added to the SDK, that supports saving and loading multichannel images, BCn, ETCn, and ASTC codecs.

Known issues and limitations

- When using GPU encoding, all source image width and height must be divisible by 4.

- GPU HW based encoding feature is only available on the Windows platform. Encoding is set only for BCn codecs and quality is limited when compared to CPU encoding.

- BC6H is not supported with GPU HW based encoding.

- KTX2 file formats for ATIn and DXT5 swizzled formats are not supported.

- ATI1n, ATI2n processed images save as BC4 and BC5 formats.

- Transcoding to ARGB_16F, ARGB_32F, and ARGB_8888 image formats is supported, all other channel formats have various data issues and have been removed until a fix is available.

- Viewing glTF and OBJ models using Vulkan(TM) rendering shows blank views.

- PSNR and Image diff analysis for mismatched channel source and destination types (F16, F32, HalfFloat, and 8bit) needs improvement.

- BC6H for OpenCL is not available in this release.

- If user-requested MIP Map level generation for GPU texture sizes in the GUI and CLI applications are invalid, they will automatically be adjusted to the lowest settable limits.

- Limited CubeMap support.

6.5 V4.0

May 2020

Features

- CMP_Core GPU encoding support

    Supports GPU based encoding with OpenCL and DX11

- Analysis Views

(Beta Feature) Displays performance data for GPU and CPU based BC1 to BC7 encoders

- CLI Analysis Log File

  The "process_results.txt" logging includes GPU performance analysis data. csv file format is also available.

- Using Codec Quality Settings

  Quality settings are available for BC1, BC2 and BC3 encoders.

- Setting Global Quality Settings

  Users can override all individual destination compression settings, using a globally set value before processing.

- Make Compatible Feature

  Compressonator SDK performs auto conversions of FP16 to byte and byte to FP16 formats when encoding textures with GPU or CPU encoders, a pre-conversion of the source data is performed, into a temporary buffer which is then sent for processing, once the processing is completed the buffer is removed.

Known issues and limitations

- GPU based encoding feature is only available on the Windows platform.

- When using GPU Encoding, all source image width and height must be divisible by 4.

- BC1, BC2 and BC3 DXC Performance is slow for quality setting > 0.6

- CMP_Core for BC1,BC2,BC3,BC4,BC5 is fully functional on both OpenCL and DX11.

- CMP_Core BC7 has limited support on OpenCL, in a few cases encoding images causes GPU and CLI application to become unresponsive.

- CMP_Core BC6 for OpenCL is not completed.

- BC6 & BC7 on DX11 uses DirectX Tex shaders, CMP_Core version will be available soon.

- GPU shaders for OpenCL and DX11 are compiled at runtime when encoding a texture for the first time, all subsequent runs use compiled shaders.

- KPerf(ms) and MTx/s are not measured for Compressonator SDK CPU encoding, only measured for CMP_Core SDK HPC-CPU and GPU encoding.

- KPerf(ms) and MTx/s measurements do not match across DXC and OCL pipelines.

- If user-requested MIP Map level generation for GPU texture sizes in the GUI and CLI applications are invalid, they will automatically be adjusted to the lowest settable limits.

- See v3.2 list for additional issues and limitations.

6.6 V3.2

December 2019

Features

- New Libraries

  Several new libraries are now provided with the SDK.

  Compressonator Core Provides block level API access to updated performance and quality driven BCn codecs.

Compressonator Framework Includes Compressonator core with interfaces for multi-threading, mipmap generation, file access of images and HPC pipeline interfaces. (SPMD & GPU support is not enabled in this release)

Compressonator SDK Has been updated to support Cube Maps, MIP Map generation. External link requirement for Open EXR has been removed.

• Cube Map Support

This release previews cube map support for images that are limited to RGBA_8888 format and RGBA_F16. Support for other formats will be provided in the next major update.

• Analysis Views

A analysis table view and results output are provided for users to view test analysis results for Performance, PSNR and SSIM analysis during and after processing of image textures.

Known issues and limitations

• HPC BC7 codec on Linux platforms shows block artifacts.

• HDR Cube maps (Format ARGB_16F) files have issues in the GUI view, only the first frame and MIP Level is displayed.

• Cube map only supports a limited set of texture types (RGBA_8888 and RGBA_F16), additional format will be added in future release.

• Cube maps with .KTX as destination format is not supported.

• When transcoding signed floats with BC6H HPC on unsigned RGBA_8888 data the images will appear distorted.

• CreateCodecBuffer (nCodecBufferType) case needs to create new codec buffers for CBT_RGBA8888, CBT_BGRA8888 and CBT_ARGB8888. The fix has been patched in this release.

• In GUI, ATI2N decode with CPU is swizzled. Decode views with GPU_OpenGL, GPU_DirectX and GPU_Vulkan are correct.

• MSE calculations are based on RGB channels only, alpha channel is not included. New MSE calculations based on MipSet data format for RGBA channels will be used in next release.

Notes

• BC5 codec uses ATI2N_XY (Red & Green) channel format if you prefer Green & Red for BC5 please use ATI2N format for encoding.

• In GUI, BC4 decode views using CPU is gray scaled based on Red channel. Next release will only use Red channel views to match GPU views.
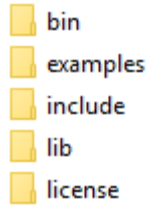
6.7 V3.1

November 2018

Features

• **Fresh New Installers:**

When installing to Windows with V3.1 release, you will notice that there are several separate new installers: SDK, GUI and CLI, which when all used result in the following file structure under \Program Files\Compressonator 3.1

bin

examples

include

lib

license

The bin and license folders are created when the user installed GUI or CLI. When the SDK is installed, an examples folder is created which contains sample application source code demonstrating the use of the Compressonator SDK. These samples use the header files from the include folder and require linking with the prebuilt sample Compressonator library provided in the lib folder. Users can also build the Compressonator library using the source code from GPUOpen. Each of these installs will not remove prior v3.0 or older installations. The user should uninstall manually using the control panel "add remove programs".

- **Texture Compression**

    BC6H: The quality of images varies slightly when processed with a CPU based encoder and decompressed by GPU compared to decompression by CPU, especially when generating MIP map levels with progressively lower resolution images. Our latest release compensates for these differences to reduce the errors that causes blocking artifacts and luminance changes.

    ETC2_RGBA and ETC2_RGBA1 is added to Compressonator SDK, that controls the degree of transparency of images for ETC2 blocks.

- **CLI Options**

    Process Source and Destination Folders:

    The new CLI adds support for processing image files from a folder, without the need to specify a file name. Using a file filter, specific files types can also be selected for compression as needed.

    CLI Analysis Log File:

    CLI will generate an output "process_results.txt" when -log is added to the compression command line options, users can change the default log file using the command -logfile, the log captures details of the source and destination files along with statistical data on performance and quality.

- **3D Model**

    Model Mesh Compression: Additional support for gltf 2.0 compression extensions (KHR_draco_mesh_compression) is added to CLI and GUI

    Selectable 3D Model Viewers: Support for the Vulkan Viewer code introduced in 3.0 can now be set in GUI application settings, this is an alpha version and not expected to work for all glTF models, feature for this viewer compared to DX12 has been limited.

    3D Model Image Diff Viewer, Auto and Manual View: This feature allows the user to select a manual override of the automated difference view of two rendered models. The Auto View switches render frames at a predetermined rate after two render cycles of each model view, the Manual View allows the users to manually switch view using the keyboards space bar.

- **Image Viewer**.

    Save View as: Users can capture viewed images to file using context menu "Save View as" to either DDS, BMP or EXR files

    Save Block as: Users can now save any block to file using "Save Source Block . . . as" where . . . is the current cursor location translated to a block position.

> **Copy to Windows Clipboard:** Users can capture images to Windows Clipboard, using keyboard keys Ctrl C (captures displayed image), Alt C (captures original source image).

Known issues and limitations

- After uninstalling the SDK or CLI, Windows short cuts for the tools folder references are not removed. When selecting them in the start menu Windows will prompt for removal.

- Vulkan Model Viewer, the code is a preview alpha version, it may cause GUI instability with various models!

6.8 V3.0

April 2018

V3.0 release will expand Compressonator assets processing from 2D to 3D. Please refer to Getting Started document (Section "Getting Started on 3D Mesh Processing") if you wish to have a quick start on V3.0.

Features

3D Mesh Optimization

- Optimize mesh (.obj model and .bin file from .glTF model) using vertex cache optimization, overdraw reduction and mesh simplification using level of details.

- Analyzing Mesh : Provide ACMR (Average Cache Miss Ratio) and ATVR (Average Transformed Vertices Ratio) statistic from mesh optimization to analyze mesh optimization only.

3D Mesh Compression

- Compress mesh (support only .obj model) using Draco library.

  Image View switch between Original and Processed

- Original Image View and Processed Image View (Compressed Image View or Pixels Difference Image View) can be switched with simple key stokes (O or P)

Known issues and limitations

- "Mesh optimization only support glTF and obj file format while mesh compression only support obj file format.

- Embedded gltf is not supported for now. Only gltf with external .bin mesh file is supported.

- .obj file texture load (with an external mtl file) and view is not supported.

- "AutoOpt" in the mesh optimization setting may not produce the best result, it is just a default setting which includes vertex cache optimization with cache size of 16, overdraw optimization with ACMR threshold 1.05 and vertex fetch optimization. The mesh optimization setting is model-dependent and depend on actual hardware. If result is not optimized, users are advised to use "UserOpt" setting and refer to Mesh Optimizer Settings for "UserOpt" selection to set each value manually or check/uncheck certain optimization.

- Mesh Optimization with all selected Mesh Optimizer Settings will need to work with mesh analyzing with the values set to match the hardware to get the optimum result.

- KTX 3 color channels multiple mip level may result in GUI crash and will be fix in next release.

6.9 V2.7

October 2017

Features

- *Linux and Mac support-* build with cmake and shell script
- Preview of 3D model viewer and asset compression
  - Compress texture images within the glTF file.
  - View 3D Models with DX12 using glTF v2.0 file format https://www.khronos.org/gltf
  - Compare compressed model textures with original

Known issues and limitations

- "Drag and drop" does not work due to User Interface Privilege Isolation technology which forbid dragging from non-admin to admin, please relaunch the app without admin again if drag and drop does not work.
- glTF viewer is pre-mature and requires DX12 feature level 12, support only Win10 RS2 or later, also Window SDK 10.0.15063.0 is recommended for best functionality of the viewer.
- Decompress view with Vulkan only support up to driver version 1.5.0. Stay tune for update to support the latest driver.
- Some glTF view might appear zoom in or out. Please use the mouse wheel to zoom in/out for better view.
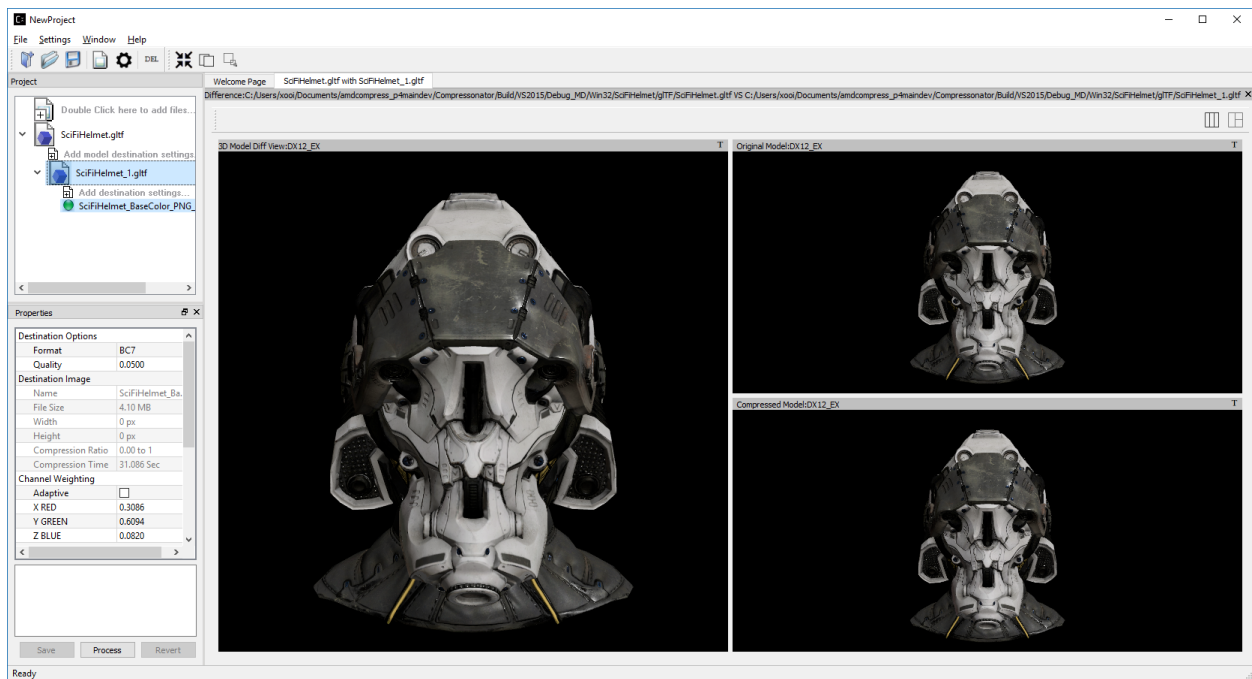
6.10 V2.5

October 2017

Features

- GUI updates includes
- Added support for HDR image view, a HDR properties window has been added to allow user adjust Exposure, Defog, Knee Low and Knee-High values.
- Added support for channel weight setting in destination setting window for compression codecs.
- Added support for alpha setting in destination setting window for BC1.

- Added option to select image view with GPU or CPU (decompress with GPU or CPU).

- GUI "Process" icon behavior has been changed for the convenience of the users:

  - "Process" icon has been enabled all the time even when users do not add any compression setting to the original images.

  - Allow user to drag and drop multiple images and click "Process" right away, in which GUI will set the compression setting chosen by the users to all original images.

- Codecs updates includes

- ASTC

- ETC2

- Improved BC6H Compression quality

6.11 V2.4

December 2016

Features

- Improved performance of ASTC compression

- Performance adjusted according to quality settings

- Fast multi-threaded implementation Example: Boat.png sample image with Quality = 0.05 and BitRate = 8.0 over 40% faster for single threaded compression compare with v2.3 35x faster for multi threaded (8 threads on a 8 Core CPU) compression

- Support MipMap generation in KTX files

- Added TGA image plugin

- Added Vulkan GPU based decompression and viewing of compressed image

- 64-bit support added to Command Line and GUI applications

Known issues and limitations

- MipMap views is not supported in GPU based decompression

Bug Fixes and Changes

- GUI application no longer requires GPUOpen CodeXL components

  - Replaced Progress Bar

  - Replaced About Box

  - Replaced Welcome Page

  - Removed Crash Reports

- To build GUI and CLI applications from source, it is required to install Qt v5.7 first (Qt v5.5 in common folder is no longer required)

- The path setting for 32 and 64 bit Qt files is set in a shared VisualStudio project file Compressonator_RootDev.proj

- Fixed the problem of corrupted BlockRate values in GUI when loading project files and processing

- Fixed the corrupted image block problem when "BlockRate" width is not equal to height during ASTC CPU based decompression

- Added check on valid ASTC user input block size (WxH) parameter

- Fixed ATC_RGB channel swizzle

- Fixed missing decompressed image output to TGA file (replaced Qt plugins with Compressonator's own TGA plugin)

6.12 V2.3

July 2016

Features

ETC2 codec for RGB textures

- Compatible with OpenGL's GL_COMPRESSED_RGB8_ETC2 API

ASTC compression & decompression of various block sizes from 4x4 to 12x12

- Supported through OpenGL APIs

- Requires GPU HW supports ASTC format

Selectable GPU or CPU based compressed image views

- GPU rendering based on OpenGL or DirectX

Channel weighting

- Enabled in Compression setting

Alpha setting enabled for BC1 (DXT1)

Contact and Support

Compressonator SDK source URL: https://github.com/GPUOpen-Tools/Compressonator

Contact Advanced Micro Devices, Inc. One AMD Place P.O. Box 3453 Sunnyvale, CA, 94088-3453 Phone: +1.408.749.4000

License

The contents of this document are provided in connection with Advanced Micro Devices, Inc. ("AMD") products. AMD makes no representations or warranties with respect to the accuracy or completeness of the contents of this publication and reserves the right to make changes to specifications and product descriptions at any time without notice. The information contained herein may be of a preliminary or advance nature and is subject to change without notice. No license, whether express, implied, arising by estoppel or otherwise, to any intellectual property rights is granted by this publication. Except as set forth in AMD's Standard Terms and Conditions of Sale, AMD assumes no liability whatsoever, and disclaims any express or implied warranty, relating to its products including, but not limited to, the implied warranty of merchantability, fitness for a particular purpose, or infringement of any intellectual property right.

AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur. AMD reserves the right to discontinue or make changes to its products at any time without notice.

FOR THE AVOIDANCE OF DOUBT THE PROGRAM (I) IS NOT LICENSED FOR; (II) IS NOT DESIGNED FOR OR INTENDED FOR; AND (III) MAY NOT BE USED FOR; ANY MISSION CRITICAL APPLICATIONS SUCH AS, BUT NOT LIMITED TO OPERATION OF NUCLEAR OR HEALTHCARE COMPUTER SYSTEMS AND/OR NETWORKS, AIRCRAFT OR TRAIN CONTROL AND/OR COMMUNICATION SYSTEMS OR ANY OTHER COMPUTER SYSTEMS AND/OR NETWORKS OR CONTROL AND/OR COMMUNICATION SYSTEMS ALL IN WHICH CASE THE FAILURE OF THE PROGRAM COULD LEAD TO DEATH, PERSONAL INJURY, OR SEVERE PHYSICAL, MATERIAL OR ENVIRONMENTAL DAMAGE. YOUR RIGHTS UNDER THIS LICENSE WILL TERMINATE AUTOMATICALLY AND IMMEDIATELY WITHOUT NOTICE IF YOU FAIL TO COMPLY WITH THIS PARAGRAPH.

Copyright and Trademarks

FireStream, and combinations thereof are trade- marks of Advanced Micro Devices, Inc. OpenCL and the OpenCL logo are trade- marks of Apple Inc. used by permission by Khronos. Other names are for informational purposes only and may be trademarks of their respective owners.

CHAPTER 9

Bibliography

Reference(1) : TEXTURE COMPRESSION TECHNIQUES Portions of this documenation is provided with permission from the authors, T. Paltashev (United States of America) and I. Perminov (Russian Federation).